PRACTICAL AUTHENTICATION IN LARGE-SCALE INTERNET APPLICATIONS

A Thesis
Presented to
The Academic Faculty

by

Italo Dacosta

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
August 2012

PRACTICAL AUTHENTICATION IN LARGE-SCALE INTERNET APPLICATIONS

Approved by:

Professor Mustaque Ahamad,
Co-Advisor
School of Computer Science
Georgia Institute of Technology

Professor Alexandra Boldyreva
School of Computer Science
Georgia Institute of Technology

Professor Patrick Traynor, Co-Advisor
School of Computer Science
Georgia Institute of Technology

Professor Raheem A. Beyah
School of Electrical & Computer Engineering
Georgia Institute of Technology

Professor Jon Giffin
School of Computer Science
Georgia Institute of Technology

Date Approved: 1 June 2012

Dedicated to my mother, Belinda Petrocelli, and the memory of my dad, Dimas Dacosta. And to the person who brings love, happiness and balance to my life, my dearest Oksana.

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the help, assistance and advice of many people. I would like to acknowledge the support I have received from the Fulbright program, Georgia Tech, my advisors, peers, friends and family.

My dream of studying in the US came true thanks to the Fulbright scholarship. I am grateful for all the help and support provided by the Fulbright program, the Institute of International Education (IIE) and the BARSA scholarship program. Specifically, I want to thank Mrs. Belsi Medina and Mrs. Dalys DeGracia in the US consulate in Panama for guiding me and other Panamanians through this process.

Since my acceptance in this program I have benefited from the resources provided by Georgia Tech. I want to express my gratitude to all the academic and administrative staff that make this university successful. I am particularly grateful to Mrs. Alfreda Barrow and Mrs. Mary Claire Thompson at GTISC for their help with all the administrative inquiries.

Most importantly, I want to thank my academic advisors and mentors. Dr. Mustaque Ahamad and Dr. Patrick Traynor who supported and guided my dissertation work while giving me freedom and flexibility to explore and develop my thinking. I greatly appreciate that over the years they have challenged my ideas, honed my research skills, provided critical feedback and inspired me to go above and beyond the boundaries of my field. I also want to acknowledge the support of my dissertation committee members: Dr. Alexandra Boldyreva, Dr. Jon Giffin and Dr. Raheem Beyah. I appreciate their timely feedback and suggestions to improve my work. In addition, I extend my gratitude to Dr. Julio Escobar, my undergraduate mentor. Dr. Escobar introduced me to the field of Information Security, encouraged and inspired

me to pursue graduate studies, and has served as a role model.

My Georgia Tech experience has been enhanced through interactions and collaborations I have had with my peers and friends. At CISEC and GTISC, I want to thank Vijay, Chaitrali, Saurabh, Hank, Frank, Jeff King, Mike Hunter, Kapil and Daisuke. I have enjoyed getting to know each of you and sharing our experiences at Georgia Tech. Outside my program, I want to acknowledge my friends in Panama and the US: Guillermo, Jimena, Marta, Nuvia, Mayli, Claudina, Claris, Min, Antonia, Ignacio, Gaspar, Jean Carlos, Roxana, Alexis and Carlos. In particular, I am indebted to Mimi and Flaviu Hodis and Jeff and Caroline Thompson, whose friendship has been invaluable during these years.

Finally, I want to thank my family for their love, assistance and encouragement. A heartfelt thank you goes to my parents, brothers, grandmother, aunts and uncles. Especially, I want to acknowledge my mother who always respected and supported my decisions and plans. Most of all, I want to express my gratitude to the most important person in my life – my wife. Her love, care and help have been fundamental to my success. Thank you my dearest Oksana, we did it our way...

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Internet applications have experienced a rapid and massive growth in popularity in the last decade. For example, today many Web and Voice over IP (VoIP) applications rely on large and highly-distributed infrastructures to process requests from millions of users in a timely manner. Due to their unprecedented requirements, these *large-scale Internet applications* have often sacrificed security for other goals such as performance, scalability and availability. As a result, these applications have typically preferred weaker but more efficient security mechanisms in their infrastructures.

Authentication mechanisms, a security layer required by most Internet applications, are an example of this trend. Mechanisms such as Digest authentication, HTTP Cookies, HTML form-based authentication and SSL/TLS server authentication are widely deployed regardless of their known weaknesses. However, as recent incidents have demonstrated, due to the increasing importance of large-scale Internet applications, powerful adversaries are now targeting and exploiting the weaknesses in these authentication mechanisms. While more robust authentication mechanisms have been proposed, most of them fail to address the specific requirements and threat model of large-scale Internet applications and, as a result, they have not been widely deployed.

In this dissertation we demonstrate that by taking into account the specific requirements and threat model of large-scale Internet applications we can design authentication protocols for such applications that are not only more robust but also have low impact on performance, scalability and existing infrastructure. In particular, we show that there is no inherent conflict between stronger authentication and other system goals.

This dissertation makes four major contributions. First, through an extensive experimental study, we demonstrate how even a simple authentication mechanism such as SIP Digest authentication can significantly impact the performance and scalability of a carrier-scale VoIP infrastructure. Second, we propose Proxychain, a SIP authentication protocol that not only provides better security guarantees than Digest authentication but also improved performance and scalability for highly-distributed VoIP environments. Third, we develop One-Time Cookies (OTC), a more secure alternative to the use of HTTP cookies as session authentication tokens. OTC is inherently robust against session hijacking attacks while preserving the efficiency and statelessness benefits of cookies. Fourth, we present Direct Validation of SSL/TLS Certificates (DVCert), a practical mechanism that offers more robust validation of SSL/TLS server certificates without requiring external third-parties or additional infrastructure. By providing stronger server authentication, DVCert effectively reduces the risk of man-in-the-middle attacks against SSL/TLS connections. In so doing, we provide robust and practical authentication mechanisms that can improve the overall security of large-scale VoIP and Web applications.

CHAPTER I

INTRODUCTION

Internet applications have experienced a rapid and massive growth in popularity in the last decade. Today, Voice over IP (VoIP) providers such as Vonage and AT&T and Web applications such as Facebook, Google and Twitter offer their services to millions of users located in different geographical areas. This vast and distributed user base generates a significantly large number of requests that need to be served in a timely fashion to provide adequate user experience. To process this high request load, these applications rely on a highly distributed and complex infrastructure composed of thousands of servers and other network components. As a result, the architects and developers of these *large-scale Internet applications* have made availability, performance and scalability their top priorities when designing and implementing such systems.

Due to this focus on performance and scalability, security has been often considered a secondary goal. In general, it is often assumed that robust security mechanisms conflict with high performance and scalability demands. Therefore, many large-scale Internet applications have deployed weaker but simpler and more efficient security mechanisms to avoid degrading other system goals. One of the reasons for this approach is that many of the mechanisms currently available were designed decades ago, when the scale and threat models of Internet applications were different. As a result, such mechanisms are not appropriate for today's large-scale Internet applications. In addition, robust security mechanisms can be more complex and expensive to deploy (e.g., additional infrastructure requirements). For example, as a response to the increasing number of attacks, Google enabled full SSL/TLS support by default

in its applications. However, while Google reported that no additional hardware was required, this project took several years and required multiple changes to servers and the SSL/TLS software stack [116]. In contrast, Facebook has enabled full SSL/TLS support as optional only, while Yahoo has not deployed it yet. To compensate for the performance impact of more robust security mechanisms, Internet applications could rely on the high capacity and elasticity of cloud computing technologies. However, such approach requires additional financial investments. Thus, it is desirable to have robust security mechanisms that are easier to deploy by using existing infrastructure more efficiently.

Authentication, the correct validation of the identity of the participants in a communication channel, is a core mechanism for most Internet applications. Without reliable user authentication, other mechanisms such as session management, access control, audit logs and billing cannot work properly. Even worse, adversaries can have arbitrary access to users' accounts and their data. In addition, reliable server authentication is important to prevent a variety of active attacks such as man-in-the-middle (MITM) and server spoofing. Unfortunately, authentication mechanisms such as Digest authentication, HTTP Cookies, HTML form-based authentication and SSL/TLS server authentication are widely deployed despite their known weaknesses. The security community has long been critical of the lack of robustness of such mechanisms and their vulnerabilities to known attacks. Nevertheless, many large-scale Internet applications rely on them due to their simplicity, efficiency and easy deployment.

The growing importance of Internet applications has increased their risk to attacks and, by extension, their need for stronger authentication mechanisms. Today's applications process a vast amount of confidential information (e.g., personal, business and government data). In addition, some applications have also become an important communication medium in countries with oppressive governments [89]. Consequently,

Internet applications have become a valuable target for a variety of adversaries, including organized crime and governments. For instance, the Open Web Application Security Project (OWASP) Top Ten report classifies "Broken Authentication and Session Management" as the third top security risk for Web applications. Recent attacks against Gmail [35], Hotmail [27] and Facebook [55], where adversaries exploited weaknesses in authentication mechanisms, are examples of this problem. Similarly, attacks against providers of authentication technologies such as RSA [83] and certification authorities [82, 36] show that adversaries are targeting different elements of the authentication infrastructure.

Large-scale Internet applications require stronger authentication mechanisms to defend against powerful adversaries. However, as mentioned earlier, while more robust authentication mechanisms are available, most of them are not appropriate for large-scale Internet applications due to their negative impact on performance and scalability and their significant deployment costs. Thus, *robust authentication mechanisms designed to accommodate the specific performance, scalability and deployability requirements of large-scale Internet applications are currently lacking.*

1.1 Thesis Statement

This dissertation aims to study the specific characteristics and requirements of large-scale VoIP and Web applications and to use this knowledge to design and implement robust authentication mechanisms for this class of applications. We argue that, by taking into account factors such as network latency, server state requirements, response times, CPU utilization and deployment costs, we can design practical authentication protocols that offer a more balanced trade-off among security, performance, scalability and deployability. Our purpose is to demonstrate that there is no inherent conflict between robust authentication and other system goals. Therefore, we propose the following thesis statement:

Current authentication protocols do not address the specific requirements and security needs of large-scale Internet applications. Authentication protocols designed for the scale and threat model of such applications can offer stronger security guarantees with low impact on performance, scalability and changes to existing infrastructure.

1.2 Research Challenges and Methodology

There are several challenges associated with the analysis of large-scale Internet applications. First, our evaluation requires the proper generation of the operational conditions of large-scale Internet applications. This includes the use of high-capacity servers, generation of high request loads and the configuration of the different infrastructure's components. Thus, several assumptions will be required due to the lack of public information regarding the infrastructure and operational conditions of large-scale Internet applications. Second, our analysis requires the correct characterization of the systems under study using different configurations and requests loads. The test load generated should be high enough to reach the maximum system's capacity (i.e., saturation). Third, our proposed solutions should include the implementation of new authentication mechanisms into existing Internet applications without affecting existing functionality or violating the application's constraints. Finally, we need to validate the security properties of the mechanisms proposed. This includes informal evaluation-by-inspection and automated formal approaches.

In general, our study uses the following methodology. First, we design and implement experimental testbeds that simulate the operational conditions of large-scale VoIP and Web applications. Second, we use these testbeds to evaluate the performance and scalability of widely supported authentication mechanisms in these systems. For VoIP, we evaluate SIP Digest authentication; for Web applications, we evaluate session authentication based on HTTP cookies and server authentication based on SSL/TLS certificates. In this context, *performance* refers to how efficiently

the application uses its resources (e.g., server CPU utilization, network bandwidth, etc.) to process requests and the maximum load supported under operational conditions. Similarly, *scalability* refers to the ability of the application to expand its capacity to handle expected or unexpected increases in the number of requests. Third, based on the results of the experimental analysis, we identify key properties that a robust authentication mechanism requires to address the requirements of large-scale Internet applications. Fourth, we use these properties to design and implement new authentication mechanisms. Our design also considers *deployability*, the effort and cost required to deploy the mechanism into existent infrastructure (e.g., modification of client and server components, additional hardware, etc.). Fifth, we implement the proposed authentication mechanisms and evaluate their performance and scalability. We then use the experimental results to compare our proposed mechanism to currently deployed mechanisms. Sixth, we also evaluate the security properties of the proposed mechanisms using informal and semi-formal security analysis based on automatic protocol verifiers. Finally, we make our prototype implementations freely available to the research community for further analysis and to extend research in this area.

1.3 Contributions

This dissertation offers the following main contributions:

- We present an detailed experimental study of the impact of SIP Digest authentication on the performance and scalability of a carrier-scale VoIP infrastructure [49, 47]. This study demonstrates how a simple protocol such as SIP Digest authentication can significantly degrade call throughput in a highly distributed scenario and explores different techniques to improve performance in such scenario.

- We propose Proxychain [51], a SIP authentication protocol that uses temporary

authentication credentials based on hash chains to not only provide better security guarantees than SIP Digest authentication but also improved performance and scalability in a carrier-scale VoIP infrastructure.

- We develop One-Time Cookies (OTC) [50], a more secure alternative to the use of HTTP cookies as session authentication tokens. OTC is inherently robust against active attacks such as session hijacking while preserving the efficiency and statelessness of HTTP cookies.

- We propose Direct Validation of SSL/TLS Certificates (DVCert) [48], a practical mechanism that offers more robust validation of SSL/TLS server certificates without requiring external third-parties or additional infrastructure. DVCert relies on existing user authentication credentials to provide stronger server authentication and effectively reduces the risk of MITM attacks against SSL/TLS.

1.4 Dissertation Outline

This dissertation is organized as follows:

Chapter 2 offers background concepts associated with large-scale Internet applications and authentication mechanisms; Chapter 3 presents relevant work in the areas of VoIP and Web authentication mechanisms; Chapter 4 describes our experimental study of the impact of SIP Digest authentication on the performance and scalability of a distributed VoIP infrastructure and our analysis of different techniques to reduce such impact; Chapter 5 presents the design and evaluation of Proxychain, a more secure alternative to SIP Digest authentication that also provides improved performance and scalability; Chapter 6 describes One-Time Cookies, a more robust session authentication mechanism that is inherently secure against session hijacking attacks and maintains the performance benefits of authentication cookies; Chapter 7 explores the problems associated with the current CA-based trust model for SSL/TLS and presents Direct Validation of SSL/TLS Certificates (DVCert), a easy to deploy

mechanism that allows more robust SSL/TLS server certificate validation without additional third-parties, preventing MITM attacks against SSL/TLS connections; Chapter 8 presents our concluding remarks and future work.

CHAPTER II

BACKGROUND

2.1 Large-Scale Internet Applications

Internet applications are a type of distributed programs that communicate using Internet protocols and services. Their process-to-process communications are performed by protocols and methods defined at the application layer of the Internet protocol suite (TCP/IP) and the Open System Interconnection (OSI) model. In general, these applications use a centralized client-server model where hosts running client programs send requests to hosts running server processes that prepare corresponding responses. However, the peer-to-peer model, where hosts act as both a client and a server and there is no a central infrastructure, is also common.

First Internet applications such as email and FTP were designed in the late 1960's and early 1970's as part of the ARPAnet, a predecessor of the Internet. With the standardization of the TCP/IP protocol in the 1980's and the ARPAnet transition to research and educational networks in the early 1990's (i.e., the early Internet) more applications were developed. In addition, several authentication mechanisms, that are still in use today (e.g., Kerberos, PAKE, PKI, SASL and HTTP Basic and Digest authentication), were designed during this period. However, the size of the Internet by mid 1990's was relatively small compared to today's Internet and, as a result, most Internet applications had small to moderate performance and scalability requirements. Moreover, the number of security threats and adversaries affecting Internet applications was also small. This scenario changed rapidly in the mid 1990's with the introduction of the World Wide Web (i.e., the Web), the commercialization of the Internet and improvements in network technologies and infrastructures. These

factors fueled the extraordinary increase in the number of hosts, users and applications, leading to the Internet we know today. Current Internet applications are not only capable of providing most of the functionalities found in traditional standalone applications but also new and advanced functionality and capabilities (e.g., electronic commerce, social networking and online gaming) that have changed our lives as well as the world's social and economical landscape.

The increasing popularity of the Internet has allowed some applications to grow to unprecedented level, resulting in *large-scale Internet applications*. Examples of such applications are search engines such as Google, Bing and Yahoo; social networking services such as Facebook and Twitter; VoIP providers such as Skype, Vonage and AT&T; and electronic commerce sites such as Amazon and Ebay. Due to their popularity, large-scale applications serve a *vast number of users*, typically in the order of millions, located in different geographical areas. As a result, these applications require a massive infrastructure capable of processing the *high request load* generated by users. For example, Ebay had an estimated of 212 millions users generating around 1 billion page views (i.e., requests) per day in 2006 [170]. To process such load, Ebay used approximately 15,000 application servers. More recently, Facebook announced that, in March 2012, it had approximately 901 million users generating close to 3.2 billion "likes" and comments request per day [91]. While Facebook does not disclose the number of servers in its infrastructure, this number was estimated to be at least 60,000 in 2010 [133]. To provide better capacity and response times, large-scale Internet applications rely on a *highly distributed infrastructure*, with server clusters located geographically close to the users. As a result, there are *high network latencies* among different components of the application's infrastructure. The high network latency makes certain requests and state synchronization operations among the application's servers considerably expensive. Moreover, large-scale Internet applications require a *scalable architecture* capable of handling expected and unexpected

increases in request load (e.g., flash crowds, emergency situations) that could affect the availability and quality of the services offered.

The specific requirements of large-scale Internet applications, described earlier, determine the security mechanisms that are used by these applications. In the following sections, we describe the basic concepts of two popular types of Internet applications: VoIP and Web applications, present example scenarios of large-scale deployments of such applications and describe the standard authentication mechanisms used by these systems.

2.2 VoIP Applications

Voice over IP (VoIP) can be defined as the set of protocols, technologies and infrastructure required to transmit voice signals over an IP network such as the Internet. VoIP has fundamentally reshaped the telephony landscape. Instead of using dedicated, circuit-switched lines, VoIP allows for phone calls to be multiplexed with other data traffic over the Internet. This convergence between voice and data communications provides a number of benefits. For instance, providers can now offer a range of new services such as video calls, video conferences and presence.

VoIP uses different protocols for signaling (i.e., session establishment and management) and data transmission. In general, there are two main signaling protocols for VoIP applications: H.323 [99] and the Session Initiation Protocol (SIP) [161]. However, SIP has more widespread support due to is simpler and more efficient design. As a result, SIP can be considered the de facto signaling protocol for VoIP. Once a session has been established, VoIP uses additional protocols such as the Real-Time Transport protocol (RTP) [166] exchange voice and video data among the participants in the session.

Authentication of user requests and server responses is performed at the VoIP signaling layer (authentication is not performed at the data layer). Thus, we will

focus on the SIP protocol and its authentication mechanisms.

2.2.1 Session Initiation Protocol (SIP)

As stated before, SIP is an application-layer signaling protocol. Instead of simply passing content between parties, SIP allows endpoints to negotiate the characteristics and protocols of communication and establish and tear down sessions [161]. SIP is used mainly to establish client-to-client sessions, instead of just client-server interactions (e.g., HTTP protocol). While widely used to perform the signaling operations for multimedia applications including Voice over IP (VoIP) and video conferencing, SIP can also be used to establish connections for applications such as instant messaging.

SIP uses uniform resource identifiers (URI) to identify each network resource. SIP URIs are similar to email addresses, following the *username@domainname* format.

A SIP network contains a number of different components:

- *User Agents (UAs)*: Instead of traditional telephones, SIP users communicate via UAs – software entities running on the user platform (e.g., desktops, laptops, SIP phones). UAs are simply communications endpoints and are responsible for initiating, responding to and terminating calls between other endpoints. A UA acts as a client (UAC) if it send a SIP requests and as a server (UAS) when it receives a requests and returns a SIP response.

- *Registrar*: To allow the mobility of users and their UAs between different domains, SIP networks operate Registrars. Like Home Agents (HAs) in Mobile IP or Home Location Registers (HLRs) in cellular telephony, SIP Registrars keep track of the current location (IP address) of a UA and are queried during call setup.

- *Proxy*: When a UA attempts to make a call, it relies on a proxy to route its request to the appropriate domain. Thus, a proxy acts as a UAC and UAS to

make requests on behalf of other clients. Proxies also assist in authentication and billing operations. Accordingly, the performance of these servers is of critical importance to providers. Because both proxy and registrar functionality is implemented in software, these tasks are often executed on a single physical node.

- *Redirect*: A redirect server is used to redirect a UAC request to the targeted UAS. Instead of forwarding requests on behalf of the UAC (like a proxy), a redirect server informs the UAC of the address of the targeted UAS. With this information, the UAC can send a request to the UAS directly.

The communication among UACs, UAS and other SIP components is based on different type of request (client to server) and response (server to client) messages. There are six type of request messages:

- *REGISTER*: used by a UA to notify its current location (i.e., IP address) to a Registrar server.
- *INVITE*: used to invite another UA to communicate and then establish a session between them.
- *ACK*: used to acknowledge a reliable message exchange.
- *CANCEL*: used to terminate a pending request.
- *BYE*: used to terminate an existent session.
- *OPTIONS*: used to query for the capabilities of SIP servers or other UAs.

The SIP responses are group into six categories that indicate the status of the current request:

- Informational (1xx): the request has been received and it is being processed.

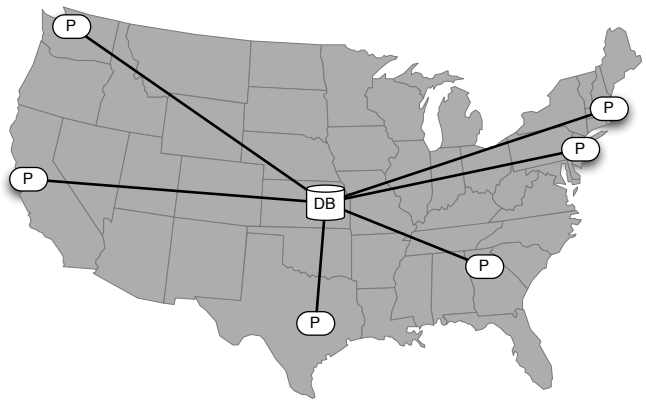


Figure 1: A hypothetical nationwide VoIP SIP infrastructure. As it is done by some cellular providers, the authentication service/database (DB) is centrally located, with proxies (P) distributed across the country.

- Success (2xx): the request was successfully received and accepted.
- Redirection (3xx): the request requires further actions to be completed.
- Client error (4xx): the server cannot process the request because it contains errors.
- Server error (5xx): the request was successfully received but the server cannot process it.
- Global failure (6xx): the request was received but it cannot be processed by any server.

2.2.2 A Nationwide SIP Infrastructure

Telephony networks have long relied on a collection of distributed databases and proxies to store user information and implement authentication. However, advances in processor speeds and ease of management have prompted a number of cellular [139] and VoIP providers such as Skype to rely on a central authentication service[1]. The use of a central database or network authentication service provides several benefits:

[1]Note that calls are placed through "Super-Nodes" in Skype, but that users sign on through a server located at 80.160.91.11.

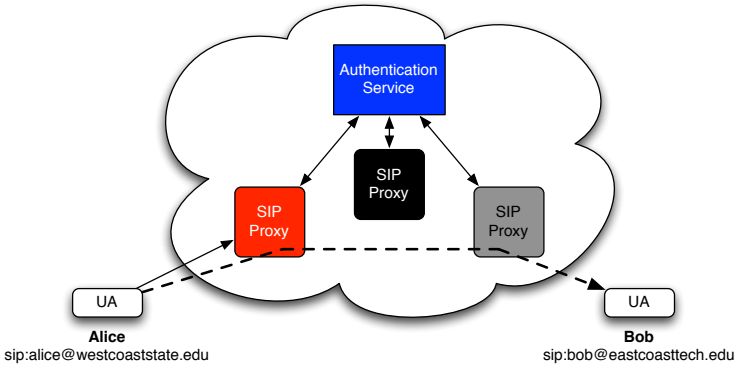


Figure 2: A different view of a nationwide VoIP SIP infrastructure. Alice and Bob communicate via a SIP provider with proxies distributed across the United States.

it simplifies management, avoids complex database synchronization operations, requires less hardware resources and allows better protection of user information (e.g., information is stored in a single location). Figure 1 provides a high-level view of a hypothetical nationwide VoIP SIP infrastructure. Notice that the provider has deployed multiple SIP proxies (P) across the country to minimize the latency associated with servicing user requests. In addition, the provider relies upon a centralized service/database to authenticate incoming call requests. The centralized service/database is located somewhere in the middle of the country to reduce the network latency with the proxies. Figure 2 shows a different view of the nationwide SIP infrastructure. A user Alice (`alice@westcoastu.edu`) in California attempts to call her friend Bob (`bob@eastcoaststate.edu`) in Georgia. Alice's call request (INVITE) is first transmitted to the closest proxy. The proxy then authenticates Alice's identity with the assistance of the centralized service/database. Successfully authenticated call requests are forwarded to the proxy currently serving Bob. After the call is established, Alice and Bob directly exchange audio packets using a protocol such as RTP.

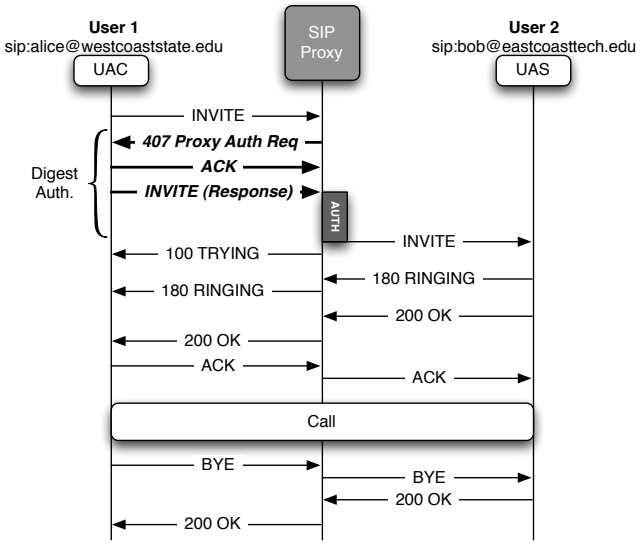


Figure 3: SIP call setup using Digest authentication (bold).

2.2.3 SIP Digest Authentication

There are a number of ways in which SIP transactions can be authenticated. The SIP standard, RFC 3261 [161], recommends strong security mechanisms such as TLS, S/MIME and IPSec to authenticate and protect SIP sessions. However, these mechanisms are complex to deploy and impose a significant overhead. Instead, most SIP providers use SIP Digest authentication, a simpler and lighter-weight challenge-response authentication protocol based on HTTP Digest authentication [71]. Furthermore, it is the only authentication protocol required in the UAs according to RFC 3261 (support for other protocols is not required). Thus, digest authentication is often considered the de facto authentication mechanism for SIP.

Digest authentication is used by SIP proxies to validate the identity of requests received from UAs (i.e., user authentication). It allows users to prove their knowledge of a shared secret (e.g., password) to a server without sending the secret unprotected over the network (protection against eavesdropping attacks).

Figure 3 shows a SIP call dialog using Digest authentication. As in most deployments, only INVITE requests require authentication. First, Alice's UA sends an

INVITE request to the proxy. The proxy determines that the request requires authentication and responds with a SIP 407 response ("`Proxy Authentication Required`") containing a nonce. Alice's UA acknowledges the reception of the challenge, computes the hash of the shared secret and the nonce and sends it back to the proxy using a new INVITE message. The proxy then computes the answer after querying a database that stores the user's shared secret. Finally, the proxy compares both values and, if they match, forwards the INVITE to the destination and the SIP dialog continues its standard flow.

Digest authentication efficiency relies on the use of hash operations and nonces, instead of symmetric or public key cryptography. In its basic form, a Digest authentication response is computed as follows:

$$response \quad = \quad MD5(HA1|n|nc|cnonce|qop|HA2)$$

where the MD5 hash algorithm is run over the concatenation of a value HA1, the nonce generated by the proxy, an optional nonce count (nc) value to prevent replay attacks, an optional client nonce ($cnonce$), the Quality of Protection (qop) value that should be set to "auth" and a value HA2. HA1 and HA2 are calculated as follows:

$$\begin{aligned} HA1 &= MD5(username|realm|password) \\ HA2 &= MD5(method|digestURI) \end{aligned}$$

where $username$ and $password$ are unique values to Alice (and ideally all other users), $realm$ is the protection domain (i.e., the provider or proxy's name), $method$ is the SIP action being authenticated and $digestURI$ is the destination address the client is attempting to reach.

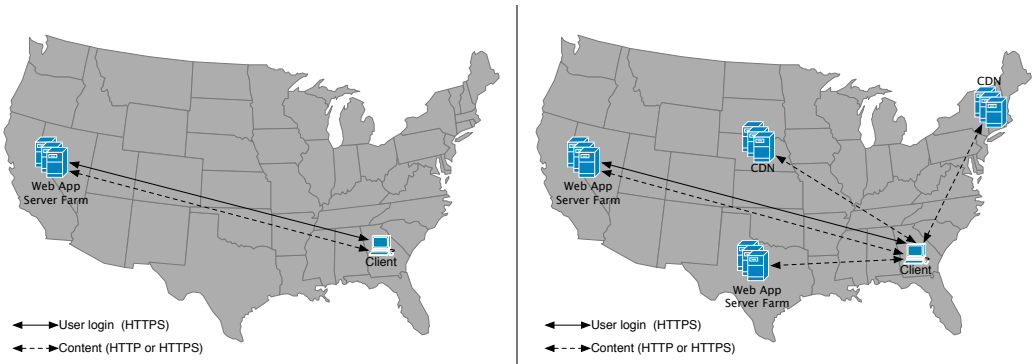
Digest authentication has a number of weaknesses. Most significantly, it only provides user authentication (i.e., no mutual authentication) and it is vulnerable to

offline dictionary attacks due to the use of low-entropy secrets (i.e., passwords) to compute the UA response. Moreover, a number of weaknesses in MD5 [195, 20, 108] have led the security community to recommend the use of other hashing algorithms. Still, in spite of its weaknesses, MD5-based digest authentication is widely supported by most SIP providers.

2.3 Web Applications

The World Wide Web (WWW), or simply the Web, is a repository of documents (web pages) linked together that are accessible over the Internet. The HTTP protocol is the main mechanisms to access these linked documents. The Web uses a client-server model where a web browser (client) is used to retrieve and display web pages from a web server. The set of related web pages and other documents stored in a web server is known as a web site.

During the early days of the Internet and the Web, most web sites only provided static content and basic functionality. Browsers were mainly used to request and display static web content. Thus, the flow of information was mostly unidirectional (i.e., server to client). However, due to the popularity of the Web, improved Internet access and the development of new web technologies (i.e., server-side and client-side code, Flash, AJAX,etc.), many web sites are now web applications, offering dynamic and customized content and a wide variety of services. In addition, the information flow is bidirectional: from the server to the client and the client to the server (i.e., user generated content). The set of new web technologies and web applications has been informally named as Web 2.0. Today's web applications not only allow novel services such as social networking, online banking and online shopping; but also enable many of the tasks traditionally performed by standalone applications (e.g., word processing, spreadsheets, video games, etc.). This trend continues as web applications begin to merge with applications in mobile devices such as tablets and smartphones (i.e.,



(a) Traditional model (b) New model

Figure 4: Simplified view of the traditional and the new Web 2.0 client-server models. In the new model, state synchronization among the web application's servers is expensive due to network latency.

mobile applications).

2.3.1 Large-Scale Scenario

Figure 4a shows a simplified view of the traditional model used by web applications. Clients located in different geographical areas send requests and retrieve content from a set of web application's servers located in a single physical location. In this model, synchronizing state among the servers is typically not expensive. However, highly distributed Web 2.0 applications have replaced the traditional model with the one depicted in Figure 4b. In the new model, a web application has data centers in diverse locations and relies on Content Delivery Networks (CDN). Clients send requests and retrieve content from servers located in different geographical areas, typically closer to the client. This approach provides several benefits such as better redundancy, improved access bandwidth and reduced access latency. In this model synchronizing state among all the web application's servers is expensive. For example, the web application cannot guarantee that all the servers have the same state information at a given moment. If this state information is required for authenticating requests, then some requests are likely to be denied wrongly.

2.3.2 The HTTP Protocol

The HyperText Transfer Protocol (HTTP) [68] is an application-layer protocol used to access Web resources. HTTP uses a messaged-based, client-server model in which browsers send requests and the servers return response messages. The server' responses include the requested resource (if successful) or a error message indicating why the request failed. HTTP is also a stateless protocol. Requests to a web server are treated as independent transactions with no relation to each other. In addition, HTTP relies on Uniform Resource Locators (URL) to identify and locate any resource on the Web. Each web resource is assigned a unique URL which defines four things: protocol, host computer, port and path (e.g., http://www.example.com/index.html).

An HTTP transaction consist of a browser request and a server response. All the HTTP requests and responses have one or more headers containing connection details and an optional message body. The first line of each HTTP request has three components: an HTTP method, the requested URL and the HTTP version being used. The HTTP method is the actual command or action to be performed on the specified resource. The HTTP standard defines several methods such as:

- *GET*: requests a resource from the web server.
- *POST*: sends information from the browser to the web server. Also used to perform an action on the web server.
- *HEAD*: similar to GET except that the web server does not return the resource requested, only the response header (i.e., no message body).
- *PUT*: sends resources from the browser to the server.
- *TRACE*: echoes the incoming request. Used for diagnostic purposes.
- *OPTIONS*: enquires about the possible methods supported by the web server for a particular URL.

However, from the list above, the GET and POST methods are the most commonly supported by web servers.

Similarly, the first line of the HTTP response messages includes a 3-digit status code and a short text description of the result of the request received. The response status codes can be divided in five groups:

- *1xx*: informational status.
- *2xx*: successful request.
- *3xx*: redirection to a different resource.
- *4xx*: the request contains an error (i.e., client error).
- *5xx*: the server encountered an error processing the request (i.e., server error).

In general, the most common status codes are: *200 OK* (the request was successful), *404 Not Found* (the resource was not found), *301 Moved Permanently* (redirects the browser permanently to a different URL) and *401 Unauthorized* (the request lacks proper authorization, HTTP authentication is required).

2.3.3 HTTPS: HTTP over SSL/TLS

Due to is simple design, HTTP offers no protection for the message's control information (headers) and payload. As a result, the Secure Sockets Layer (SSL) protocol [72] and its successor, Transport Layer Security (TLS) [53] were developed to provide confidentiality and integrity protection for HTTP transactions (and latter extended to protect other application protocols). SSL/TLS offers a transparent transport encryption layer to web applications and it is widely supported by most browsers and web servers. Thus, HTTPS (i.e., HTTP over SSL/TLS) has become the standard way to provide protection to web communications. SSL/TLS can also be used to provide client and server authentication (see Section 2.3.6).

2.3.4 User Authentication

Most web applications require users to create an account (i.e., registration) and log in to create or resume a session. During registration, users are typically required to create a password that will be used during the log in process to authenticate the user. User authentication is one of the core security mechanisms used by most web applications. If user authentication fails, then an adversary can take control of the user account and access all the data stored in the application. In this section, we describe some of the main user authentication mechanisms used by web applications.

2.3.4.1 HTTP Basic and Digest authentication

Basic and Digest authentication [71] are HTTP's built-in authentication mechanisms. Both mechanisms assume that each user shares a secret (password) with the server. The user needs to prove her knowledge of the password to the server in order to be able to access protected resources. For this purpose, Basic authentication requires the user to send her password unprotected (cleartext) over the network to the server. However, sending cleartext passwords over the network is a serious security flaw because an attacker can eavesdrop an authentication session and learn the user's password. Digest authentication avoids this problem by sending a hash of the password and other authentication parameters instead of the password itself. In this way, it is more difficult for an attacker to obtain the user's password by capturing her network traffic.

While Basic and Digest authentication were the main authentication mechanisms used during the early days of web applications, today they are rarely used in applications of medium or large complexity. One of the reasons is that these mechanisms do not integrate well with the look-and-feel of web applications (e.g., they rely on the browser's dialog boxes).

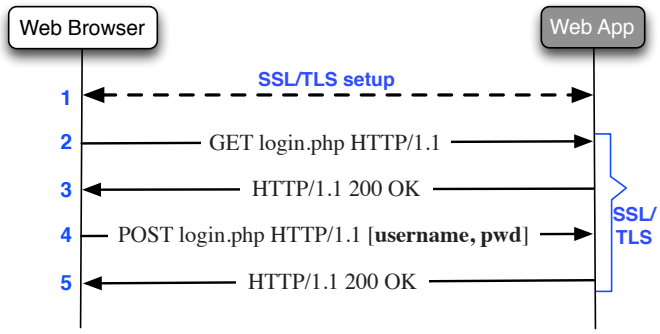


Figure 5: HTML form-based authentication example.

2.3.4.2 HTML Form-based Authentication

With the adoption of server-side scripting languages, web applications became capable of directly validating users credentials, instead of relying on server mechanims (e.g., digest authentication). Thus, HTML form-based authentication rapidly became the main user authentication mechanism used by current web applications due to its simplicity and flexibility. It has been estimated that more than 90% of web applications use this authentication method [179].

Figure 5 depicts an HTML form-based authentication flow. First, the browser and the server establish a SSL/TLS connection (step 1) to protect the subsequent HTTP transactions. Next, the browser sends a request for the web application's login page (step 2), which is then returned by the server (step 3). The login page includes an HTML form that is used to capture the users's authentication credentials (i.e., username and password). Once the user inputs her authentication information and presses the submit button, the browser sends the user's authentication credentials to the server in a POST request (step 4). Finally, the server compares the values received with values stored in the application's database. If the values match, the user is successfully authenticated and the server sends a 200 OK HTTP response to the browser (step 5). If the verification fails, the application typically asks the user to input her authentication information again (i.e., sends the login page again to the browser), for a limited number of times.

2.3.4.3 Other Authentication Mechanisms

Web applications can also rely on other well-known user authentication mechanisms such as NTLM (NT LAN Manager) and Kerberos. However, these mechanisms are rarely used in Internet scenarios (they are more appropriate for intranets and private networks). Similarly, SSL/TLS client certificates can be used to provide strong user authentication; however, they are typically used only in scenarios with high security requirements due to their associated deployment and management costs. Some web applications use multifactor authentication to provide more robust authentication. In this approach, users employ hardware tokens that generate one-time passwords. More recently, some web applications can also send one-time codes to users through secondary channels such as text messages and phone calls. Still, multifactor authentication mechanisms are not widely deployed due to their implementation costs and effects on user experience. Finally, web applications can also rely on third-party authentication services such as OpenID [146] and Microsoft Passport [132]. In this approach, users only share authentication credentials with an identity provider. To log into a web application, users authenticate first with an identity provider which vouches for the user identity to the targeted web application. This approach, however, has not been widely adopted.

2.3.5 HTTP Cookies and Session Authentication

HTTP does not provide support for session management. Each request and response exchanged between a client and a server are considered an independent transaction. While this is sufficient for most basic static pages, the need for session management mechanisms increased with the development of the first web applications. In 1994, Netscape proposed the use of HTTP cookies [113, 114, 15] for web session management. Due to their simplicity and efficiency, HTTP cookies were rapidly adopted by all major browsers and web applications, becoming the default mechanism for web

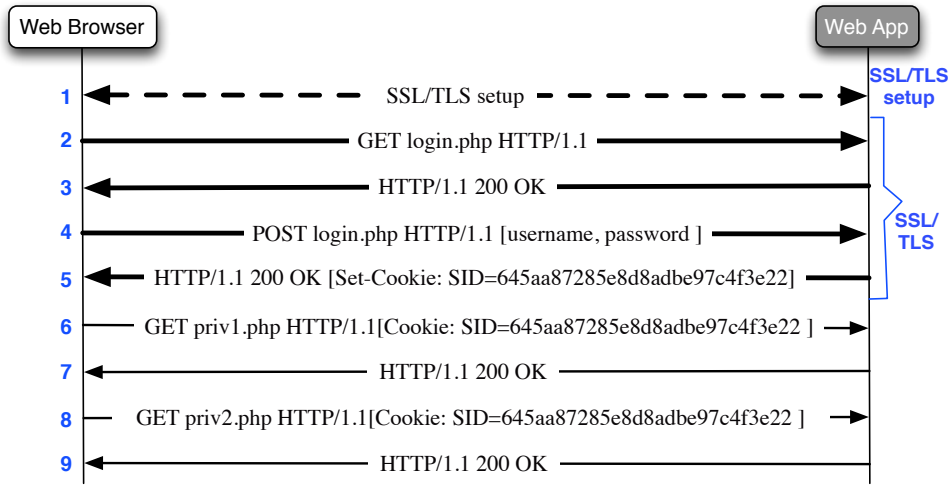


Figure 6: Session authentication using HTTP cookies. The browser attaches the same cookies to all the HTTP requests sent to a particular domain.

session management.

Cookies consist of name-value pairs containing session information that is stored in the browser. A web application generates cookies and sends them to the browser using the *Set-Cookie* HTTP response header field, as shown in the following example:

```
Set-Cookie:  SID=645aa87285e8d8adbe97c4f3e22
```

In addition to the cookie's name and value, the web application can define other attributes such as the *domain* and *path* to define the cookie's scope, the *expiration* to determine cookie's lifetime, the *httponly* flag to decide if client-side scripts can access the cookie and the *secure* flag to determine if the cookie should be transmitted only using a secure channel (i.e., HTTPS). Once cookies are accepted by and stored in the browser, they are appended to each request sent to the web application, using the *Cookie* HTTP request header field, as shown in the following example:

```
Cookie:  SID=645aa87285e8d8adbe97c4f3e22
```

2.3.5.1 Authentication Cookies

Authentication cookies are generally created during the user login process. After successful validation of the user's authentication credentials, the web application generates authentication cookies and sends them to the browser. The browser attaches these cookies to each request that requires authentication, based on the cookies' scope and flags. Once established, authentication cookies become a temporary replacement of the user's password credentials.

Figure 6 shows how cookies are set and used to authenticate browser requests. First, the browser and the server establish an SSL/TLS connection to protect the user login process (step 1). Next, the user authenticates to the web application using HTML form-based authentication (steps 2 to 5). Notice that after the successful validation of the user credentials (step 4), the server response includes the authentication cookie (SID) as part of the message header (step 5). The browser stores the cookie and proceeds to attach it to each request that match the domain and path of the cookie. For example, requests for private resources in steps 6 and 8 require authentication; thus, the browser attaches the cookie to these requests. If the cookies are valid, the server returns the requested resources (steps 7 and 9). In addition, notice that the requests and responses after user login are not protected by SSL/TLS. This is a common practice in web applications to reduce performance overheads and complexity due to SSL/TLS. However, this practice also introduces security vulnerabilities due to the static nature of the authentication cookies (see Chapter 6).

Authentication cookies should be carefully constructed to prevent abuse. However, due to the heterogeneity of web applications, there are no standards for designing and implementing cookie-based session authentication mechanisms. As a result, many web developers design and implement in-house mechanisms, frequently introducing critical vulnerabilities [73, 191]. In general, web applications rely on well-known cryptographic techniques (e.g., symmetric encryption algorithms and cryptographic hash

functions) and secret information (e.g., cryptographic keys, users' passwords) to build authentication cookies. The secret information is shared among all the web application's servers that need to verify the authenticity of users' requests. Nevertheless, while the confidentiality and integrity of cookies can be guaranteed by cryptographic mechanisms, attacks are still possible based on how cookies are used.

2.3.6 Server Authentication

Most of the protocols previously described only provide user authentication. However, implementing both user and server authentication (i.e., mutual authentication) is important to defend against different types of attacks such as server spoofing, phishing and MITM attacks (see Section 7.1.2). The most common approach to validate the identity of a web application's server is via X.509 digital certificates [4]. A digital certificate binds the server's identity (i.e., domain name) to the server's public key and it is signed by a Certification Authority (CA) trusted by both the server and the browser. CAs are required because the browser and the server do not share any secrets at the SSL/TLS layer; thus, a trusted third-party is needed to vouch for the authenticity of the server's certificate.

2.3.7 The SSL/TLS Protocols and Web Applications

Figure 7 presents a simplified view of the SSL/TLS handshake protocol between a browser and a web server. First, the browser sends a SSL/TLS *ClientHello* message to the server to request a new SSL/TLS connection (step 1). The server responds with a *ServerHello* message that typically includes the server SSL/TLS certificate *Cert* (step 2). Next, the browser proceeds to validate the server certificate to ensure it is valid (i.e., it has not expired or revoked) and that it has been signed by a trusted CA. After successful validation, the browser uses the public key included in the certificate to encrypt protocol state parameters that are required to complete the handshake (i.e., parameters required to derive the session keys). The browser

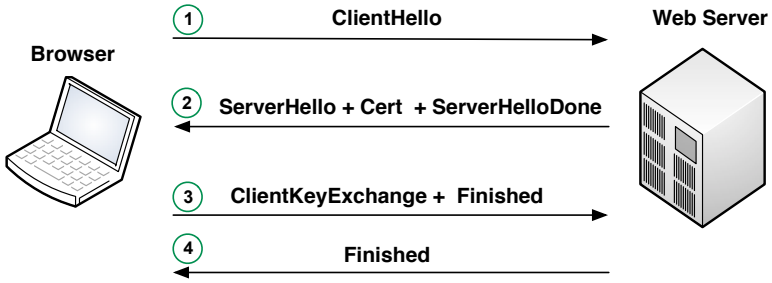


Figure 7: Simplified view of the SSL/TLS handshake protocol. The server can only complete the protocol if it has the private key corresponding to the public key in the server certificate (*Cert*. Thus, a successful handshake validates the server's identity.

then sends the encrypted information in a *ClientKeyExchange* message to the server, together with an authenticated and encrypted *Finished* message (step 3). Next, the server decrypts the information received and proceeds to use this information to generate the SSL/TLS session keys. The server then sends a *Finished* message to the browser that is encrypted and authenticated with the sessions keys (step 4). Finally, the browser uses the session keys to decrypt and to validate the server's *Finished* message. If this validation succeeds, then the server has proven to the browser it has the private key corresponding to the public key in the certificate. As a result, the browser is able to successfully verify the server's identity and the sessions keys are then used to protect the subsequent HTTP messages.

CHAPTER III

RELATED WORK

3.1 VoIP Applications

3.1.1 SIP Authentication and Its Impact on Performance

Performance and scalability are critical properties for telecommunication networks. Emergency situations (i.e., natural disasters, terrorist attacks [88]), large scale events (i.e., American Idol [188], President's Inauguration Day [159]), and Denial of Service attacks [119] are examples of events that can cripple the availability of the telecommunication network. SIP, as one of the central protocols of many IP telephony networks, plays an important role in the performance and scalability of such systems.

The performance and scalability of a SIP infrastructure has been the subject of several studies. Schulzrinne et al. [167] described a set of metrics for evaluating and benchmarking the performance of SIP proxy, redirect and registrar servers, and an architecture and techniques to measure SIP server performance. Other studies have focused exclusively on the performance of a SIP proxy, analyzing how different configurations impact overall performance. Nahum et al. [141] evaluated the impact of state management, transport protocol and authentication on proxy's performance. Their findings show that proxy's performance varies greatly depending on the configuration chosen and that authentication has the greatest impact across all the configurations analyzed. Salsano et al. [162] presented a similar study focused on security configurations such as Digest authentication and TLS. They also concluded that authentication considerably affects the performance of the proxy. Ono et al. [145] analyzed how the use of TCP affects the performance and scalability of a proxy. In addition, the author proposed several configuration changes to improve performance. Cortes et

al. [44] evaluated the parsing, string processing, memory allocation, thread overhead and overall capacity of a proxy.

Several studies have suggested modifications and mechanisms to improve the performance of SIP proxies. Janak [102] proposed the use of lazy parsing (parse only the minimum number of headers required) and incremental parsing (parse contents of headers incrementally) as significant performance optimizations for proxies. Balasubramaniyan et al. [13] proposed an algorithm to dynamically distribute call state information among proxies organized in a hierarchical SIP infrastructure. The authors demonstrated how this dynamic distribution of call state provides significant performance gains compared to standard configurations. Similarly, Cortes et al. [45] suggested that proxies belonging to an IP Multimedia Subsystem (IMS) should be configured with stateless and stateful functionality, allowing proxies to dynamically adapt to network and CPU conditions. Singh et al. [173, 174, 175] presented a reliable, scalable and interoperable Internet telephony architecture for user registration, call routing, conferencing and unified messaging using commodity hardware. The authors proposed an identifier-based two-stage load sharing method based on Web server redundancy techniques for high service availability and scalability. Finally, Shen et al. [169] proposed three new window-based feedback algorithms for overload controls in proxies and compared them to current algorithms.

Still, while some of the previous studies have analyzed the impact of SIP Digest authentication on the proxy's performance, none of them consider large-scale scenarios where the user authentication credentials are stored in a remote database or a network authentication service is used instead (i.e., RADIUS). Such scenarios are common in real deployments because they allow better scalability and easier management. Moreover, none of the previous works propose more efficient alternatives to SIP Digest authentication, even though it has been shown that Digest authentication can have a considerable impact on proxy's performance and scalability, particularly if a remote

database or authentication service is used.

Our initial work in this area [49, 47], described on Chapter 4, addressed this gap by evaluating the use of multiple processes and batch requests to improve the performance of a SIP proxy configured with Digest authentication and a remote database. The use of batch queries has been explored in other areas. For example, in file systems such as XFS [181] and FARSITE [8], batching file updates has been shown to improve performance significantly. Moreover, batch queries have also been used in business applications [125]. Boneh and Shacham [168] described how batching in the SSL handshake protocol can improve the performance of Web servers. Our work is the first to apply the concept of batch requests in a SIP proxy to improve performance.

3.1.2 Robust and Efficient SIP Authentication

SIP Digest authentication is often considered the default authentication mechanisms for SIP (see Section 2.2.3). The SIP standard, RFC 32611 [161], also recommends more robust security mechanisms such as TLS, IPsec and S/MIME that not only provide authentication but also other security guarantees (e.g., integrity and confidentiality). However, several studies have shown that these mechanisms are computationally expensive [134, 130, 41, 34] and, thus, not suitable for VoIP applications with high-performance requirements. In addition, these mechanisms are more complex to deploy and manage (e.g., client certificates are required). As a result, Digest authentication is the preferred authentication mechanism for most SIP deployments due to its efficiency and simplicity.

However, as the previous section described, several studies have shown that Digest authentication still produces a significant impact on the performance of a SIP infrastructure. For example, Nahum et al. [141] demonstrated that Digest authentication can degrade the performance of a SIP proxy by a factor of four, depending on the scenario. Similarly, Salsano et al. [162] found that the Digest authentication

accounted for nearly 80% of the processing cost of a stateless SIP proxy. Our own work [49] showed that the loss in performance is worse in distributed environments with a remote authentication database. We measured a performance drop of almost three orders of magnitude when the network latency between the proxy and the authentication database was 30 ms (see Chapter 4). While our work suggests a hybrid request mechanism to reduce the impact on performance, the central database can still become a bottleneck if the load from the proxies increases. Also, the database can be vulnerable to Denial of Service (DoS) attacks. For example, multiple malicious clients could generate enough request load to saturate the database, a type of attack that has been demonstrated practical in cellular networks by Traynor et al. [189].

The processing costs of Digest authentication also affect the level of security it can offer. For example, due to its performance overhead, Digest authentication is only used to authenticate a subset of the SIP message requests (e.g., INVITE and REGISTER messages) used in a SIP session. This practice and the lack of server-side authentication allow adversaries to execute several attacks against SIP infrastructures [207, 3]. In addition, Digest authentication is considered a weak authentication protocol by cryptographic standards. For example, it is vulnerable to offline dictionary attacks and it does not support mutual authentication. Therefore, a variety of active attacks [203, 59, 195] can be used by adversaries to compromise the security guarantees offered by Digest authentication.

The security community has proposed more robust alternatives to Digest authentication. For example Wang et al. [195] suggested enforcing TLS or IPSec between clients and proxies. However, as mentioned before, such mechanisms are more expensive than Digest authentication. Tao et al. [183] proposed an alternative authentication mechanism for SIP that relies on shared keys and symmetric encryption (e.g., AES). This mechanism provides mutual authentication and it is resistant to offline attacks, but it requires complex key establishment and management protocols (i.e.,

difficult to deploy and scale). Similarly, researchers have proposed a great variety of robust authentication mechanisms based on the Diffie-Hellman key exchange[203], the RSA problem [33] and Elliptic curve cryptography (ECC) [205, 38, 193]. While these mechanisms solve most of the weaknesses found in Digest authentication, they require more expensive cryptographic operations. Thus, their overhead is excessive for large-scale SIP deployments. In addition, most of these mechanisms lack a thorough performance evaluation. As a result, none of these alternatives have been deployed in practice and Digest authentication is still widely used.

To solve the performance and security problems of Digest authentication, we propose Proxychain [51] (see Chapter 5), a novel authentication mechanism based on temporary authentication vectors stored in the proxies. The use of temporary authentication vectors is also explored in the area of 3G cellular networks by the Authentication and Key Agreement (AKA) [2] security protocol. However, instead of using multiple authentication vectors as AKA does, Proxychain relies on a modified hash chain construction [115] to provide mutual authentication. Hash chains have been used in security protocols in different domains where efficiency is critical such as sensor networks [152, 124] and RFID tags [182]. Our work is the first to take advantage of the security, performance and space efficient properties of hash chains to reduce the overhead of the authentication process in SIP.

3.2 Web Applications

3.2.1 More Robust Alternatives for Session Authentication

As we described in Section 2.3.5, web applications rely on HTTP cookies to authenticate users' requests. However, the use of cookies as session authentication tokens has raised security concerns since their adoption in the mid-90's. Several surveys [73, 191] have shown the multiple problems affecting web authentication mechanisms, including

vulnerability to session hijacking attacks (see Section 6.1). In response to these problems, researchers have proposed changes to authentication cookies to improve their robustness. Park et al. [150] and Fu et al. [73] suggested cookie mechanisms that provide better confidentiality and integrity guarantees by using well-known cryptographic techniques. In addition, these authors proposed the use of cookie expiration times to limit the impact of session hijacking attacks. However, many applications use long expiration times to avoid affecting user experience, reducing the effectiveness of this approach. Juels et al. [104] proposed the use of cache cookies, different forms of persistent state in the browser (e.g., browser history, temporary internet files), as an alternative to cookies for storing user and session identifiers. While resistant to pharming attacks, cache cookies still need HTTPS protection to prevent active attacks. Bortz et al. [25] demonstrated a new class of attacks to steal cookies, related-domain attacks, where cookies stored by one site can be modified by another if the two sites happen to share a sufficiently long suffix. To prevent this type of attacks the authors proposed origin cookies, an extension to standard cookies that require minimal implementation costs. However, as the previous solutions, origin cookies are still vulnerable to session hijacking. Another alternative to authentication cookies is the use of hidden form fields to store authentication tokens per page. For example, the ASP.NET ViewState [136] mechanism uses this approach. Still, this approach requires additional state in the server and breaks browsing functionality (e.g., the back button). In addition, there is a small window of vulnerability where the adversary could send a valid ViewState token to the server before the user. In this case, the adversary will be able to send at least one arbitrary request that will be accepted by the server.

The problem with the previous mechanisms is that they still rely on static tokens to authenticate requests; thus, allowing session hijacking attacks. Therefore,

researchers have also explored the use of unique tokens per request to prevent arbitrary reuse. Liu et al. [123] proposed a secure cookie protocol that creates unique cookies based on a session secret and a SSL/TLS session key. While Liu's protocol has low server state requirements, it requires the use of an SSL/TLS channel (i.e., HTTPS). Moreover, browsers and web servers typically do not have access to SSL/TLS session keys, particularly when HTTPS reverse proxies are used. Blundo et al. [24] designed a lightweight mechanism for web caching authentication that relies on unique tokens based on a hash chain. While the use of a hash chain prevents session hijacking attacks, it requires additional state in the web application – a costly requirement for highly distributed web applications (see Section 2.3.1). Ben Adida proposed SessionLock [7], a session authentication protocol that also relies on a session secret to generate unique authentication tokens to prevent session hijacking. SessionLock's main novelty is the use of URL fragment identifiers to store the session secret. This approach allows SessionLock to be implemented using only JavaScript, avoiding browser modifications. However, this JavaScript-only approach makes SessionLock vulnerable to active attacks (e.g., code injection) and affects its correctness (e.g., session secrets can be exposed or lost accidentally). Moreover, SessionLock is also stateful in the web application (see Section 6.5.5 for a comparison between SessionLock and our proposed mechanism). More recently, Choi and Gouda presented HTTPI [39], a new secure transport protocol more efficient than HTTPS because it only provides server authentication and session integrity (i.e., no confidentiality). As previous solutions, HTTPI uses a session secret to generate unique authentication tokens per request. Still, it also requires state in the web application. Finally, researchers have also explored the use of capability-based web servers such as Waterken [40] to provide stronger security guarantees to web applications, including session authentication. While this approach is more robust and does not require browser modifications, it still needs significant changes of the web application (i.e.,

new web server platform).

None of the mechanisms previously described has been widely deployed. While several of them prevent session hijacking, they fail to address the requirements of highly distributed web applications, particularly requests' statelessness. Therefore, like SIP Digest authentication in VoIP deployments (see Section 3.1.2), HTTP cookies are still the preferred method for session authentication in web applications due to their simplicity, efficiency and wide support. In addition, many web applications have chosen always-on HTTPS as the main defense against session hijacking attacks. To enforce always-on HTTPS and prevent downgrading attacks (e.g., SSL stripping attacks [154]), several policy mechanisms have been developed. Jackson and Barth [100] proposed ForceHTTPS, a browser add-on that ensures that all session cookies are securely configured and forces all HTTPS errors to be treated as critical. A similar approach is used by the Electronic Frontier Foundation (EFF) tool HTTPS Everywhere [60]. Based on this idea, a new web security policy mechanism, HTTP Strict Transport Security (HSTS), is being proposed to the IETF [94]. Still, these policy mechanisms only enforce the use of HTTPS, they do not improve HTTPS deployment or performance. Moreover, even with always-on HTTPS, cookies can be disclosed through many different attack vectors (see Section 6.1). Therefore, to effectively prevent session hijacking attacks, a more robust, efficient and practical alternative to authentication cookies is needed. In Chapter 6, we describe our proposed solution.

3.2.2 Stronger Server Authentication

In the previous Section, we stated that many web applications are relying on SSL/TLS to protect HTTP authentication cookies. However, the confidentiality and integrity guarantees offered by SSL/TLS depend on the correct validation of the server's identity via SSL/TLS certificates. If server authentication fails, then adversaries will be able to defeat the protection offered by SSL/TLS by using MITM attacks (see

Section 7.1.2).

Most browsers perform multiple checks to validate SSL/TLS servers certificates and authenticate the server-side of the communication. If any of these checks fails, the browser relies on security indicators (e.g., warnings messages) to notify the user. Unfortunately, average users tend to ignore these indicators due to the lack of training and high false positive rates [52, 164, 180, 93]. More effective security indicators have been proposed [204, 202, 180], but have not been widely adopted by major browser vendors.

Still, adversaries can defeat security indicators by using *forged certificates* obtained through the exploitation of well-known weaknesses in the PKIX CA trust model [63, 90, 5]. These forged certificates are accepted by most browsers without raising any alert; thus, they are commonly used for MITM attacks [176]. As recent surveys show [57, 160, 58], the increasing complexity and lack of transparency of the CA trust model facilitates this type of attacks.

As a response to the increasing threat imposed by forged certificates, the CA/Browser forum proposed Extended Validation (EV) certificates [32] – certificates emitted under stricter identity verification procedures. However, the effectiveness of this operational measure have been questioned by several studies that show that average users do not differentiate between EV and standard certificates [101, 19]. The CA/Browser forum also published baseline standards for the secure operation of CAs [185]. Still, these operational measures do not solve the main problems of the CA trust model.

Multiple browser-based mechanisms have been proposed to detect forged certificates. For instance, browser extensions can keep track of the certificates used by the browser and can detect certificate changes [1, 201, 176]. While simple, the effectiveness of this approach is affected by false positives and lack of user training. A related technique, known as certificate pinning [66], uses a white-list of certificates for important domains that are hardcoded in the browser. This solution is less prone

to false positives; however, it is neither flexible nor scalable. A more robust approach is the use of secondary channels such as cellular networks [151] and Tor [9] to obtain additional copies of the server certificate. Assuming that adversaries have no control over the secondary channels, any inconsistency among the certificates received will indicate a possible MITM attack. Unfortunately, this technique has considerable deployment costs and can introduce significant delays to SSL/TLS connection setup.

Most research in the area of MITM defenses focuses on using additional third-parties to improve or replace the CA trust model. For example, mechanisms such as Perspectives [196], Convergence [129], VeriKey [178] and Crossbear [96] allow users to choose multiple network notaries that can complement or replace CAs signatures. In this approach, the browser queries notaries located in different network vantage points to determine if they have observed similar certificate information for a particular domain. The Mutually Endorsing CA Infrastructure (MECAI) [64] proposal also suggests the use of notaries for certificate validation. However, instead of introducing new authorities, MECAI uses existing CAs as notaries. Thus, in addition to verifying the CA signature, the browser randomly queries other CAs for additional proofs of authenticity. A different technique is presented by the Electronic Frontier Foundation (EFF) Sovereign Keys (SK) project [61]. In this project, a domain certificate includes an additional integrity signature created with the domain's sovereign key. To verify this signature, browsers can obtain the corresponding sovereign key from a semi-centralized, append-only public data structure. Google's Certificate Transparency (CT) [118] proposal also relies on a similar data structure, but instead of storing keys, it stores records of each certificate emitted by a CA. Browsers can then validate if they are using the correct certificate for a particular domain by querying this public audit log. The IETF DNS-based Authentication of Named Entities (DANE) working group [95] is developing protocols that use secure DNS (DNSSEC) extensions to bind certificates to domain names. In this approach, often considered the most

robust, certificates may have both CA and DNSSEC signatures or only the latter. Finally, while third-party based solutions offer several benefits, their adoption has been hindered by multiple problems such as deployment and operational costs, lack of user training, false positives and others (see Section 7.1.3).

To a lesser degree, researchers have also explored the use of shared secrets (e.g., passwords) to defend against MITM attacks. For example, the TLS-SRP protocol [184] uses SRP [199] for mutual authentication and SSL/TLS key derivation based on the user's password (i.e., certificates and CAs are not required). Hence, MITM attacks are not possible without knowledge of the user's password. However, TLS-SRP requires inter-layer communication between the application and the SSL/TLS stack, breaking SSL/TLS transparency. A different technique is to use shared secrets for channel binding [197, 10], as proposed in the Session Aware (TLS-SA) user authentication protocol [148]. To detect MITM attacks, TLS-SA uses authentication codes based on user credentials and SSL/TLS session information, effectively binding the application and SSL/TLS layers. TLS-SA, however, requires client certificates and hardware tokens to resist offline dictionary attacks, affecting its adoption. Finally, the Mutual Authentication Protocol for HTTP [144, 143] also combines user authentication with SSL/TLS channel binding, but it relies on the user's password instead of client certificates. To provide mutual authentication and prevent offline guessing attacks, this mechanism uses a PAKE protocol (KAM3 [97]). However, this mechanism requires additional server state, only protects the login connection and requires changes to the browser and web application login UI (a significant challenge for deploying PAKE-based protocols[65]).

MITM attacks could also be detected by uniquely identifying servers based on their inherent features and the characteristics of the network path between them and their clients (i.e., server fingerprinting). This idea has been used in IP traceback mechanisms [177, 163, 92] to identify the origin of spoofed IP packets. Moreover, this

technique can be improved by also measuring features of the application-layer payload. This idea has been used to fingerprint the source and path taken by voice calls [14]. However, the size and highly distributed nature of large-scale web applications hinder the use of such techniques for preventing MITM attacks. First, an adversary can spoof most of the TCP/IP control information due to the lack of integrity protection mechanisms. Second, network variability (e.g., network jitter, network failures) can negatively affect the accuracy of such techniques. Third, browsers typically send requests to different set of servers on each session with a large-scale web application. As a result, browsers will need an updated list of all the servers' fingerprints used by each web application. For large-scale web applications, such list could contain thousand of entries and it could require frequent updates as new servers and network paths are added or removed. A related approach is to measure the time delay introduced by an adversary during a MITM attack [12, 192]. Still, network variability is likely to affect the accuracy of such approach, resulting on high false positive rates.

In summary, while a considerable amount of research exists in this area, currently there are no effective and practical defenses against MITM attacks. Mechanisms relying on third-parties, the most popular approach, are too complex and expensive to deploy and operate at large scale. In the mean time, the number of attacks against CAs and SSL/TLS continues increasing. Therefore, it is important to find simpler and easier to deploy defenses against this threat. In Chapter 7, we present our proposed mechanisms for more robust SSL/TLS server certificate validation without external third-parties.

CHAPTER IV

IMPROVING AUTHENTICATION PERFORMANCE OF DISTRIBUTED SIP PROXIES

The *Session Initiation Protocol* (SIP) is an application layer signaling framework (see Section 2.2.1). Instead of simply connecting participants, SIP allows users to negotiate the terms of, establish and terminate sessions. Because of its relative simplicity (i.e., human readable encoding) and flexibility (i.e., transport layer agnosticism), SIP is now used by a wide array of multimedia services including video conferencing, instant messaging and presence.

SIP is also being widely adopted by IP telephony providers, including Vonage and AT&T, as the foundation of large scale deployments. Such providers typically deploy SIP proxies, nodes responsible for routing call requests and assisting in billing operations, across multiple geographic regions in order to efficiently deal with substantial volumes of traffic. With the assistance of a centralized remote server, these proxies also aid in the authentication of users and handle signaling requests. This architecture is often used to protect back-end databases from attack and to minimize distributed consistency issues. In the face of increasing volumes of both legitimate and malicious (i.e., spam) traffic at currently deployed nodes, developing effective techniques for scaling proxy throughput is becoming a critical task.

In addition, several studies have analyzed the impact of authentication, particularly Digest authentication, on the performance of a SIP proxy (see Section 3.1.1). However, these studies have focused on the evaluation of single proxy configurations with a local authentication service or database. Taking into account that large-scale VoIP deployments rely on distributed architectures, it is important to understand

how an authentication mechanism such as Digest authentication can affect the performance and scalability of this these architectures.

In this chapter, we present an empirical study of the impact of two techniques, parallel and batch requests, designed to improve the throughput of authenticated signaling requests for distributed SIP proxies sharing a remote database. Our results show that an enhanced version of the currently recommended parallel process execution method can dramatically increase proxy throughput. This improvement comes at the cost of a significant bandwidth overhead ($\approx$ 25 Mbps) and an increased dropped call rate. Request batching, an alternative technique that has not previously been applied in this domain, is then shown to greatly reduce the bandwidth costs but fails to achieve the same high throughput as the first technique. We demonstrate that a carefully balanced hybrid of these two approaches can achieve throughput at the proxy equal to the capacity of the authentication service, an improvement of 33% over the best parallel execution approach, using 77.3% and 76.6% less bandwidth for requests and responses between proxy and database and maintaining call dropping rates below 1%. We note that these improvements are not simply the result of maximizing the settings of the two mechanisms and that the careless combination of these techniques can actually degrade performance.

Through this work, we make the following key contributions:

- **Modify and extend a popular open source SIP proxy to support batch requests:** We made several enhancements to the open source OpenSER proxy software to allow for far greater throughput. We discuss our modifications and make them available as a patch to the community at: `http://www.cc.gatech.edu/~idacosta/proxy_batch.html`

- **Characterize performance improvements of both parallelization and batching:** We performed an extensive measurement study of the performance

gains realized by the use of multiple proxy processes, the use of batching and the range of combinations of these two approaches. Our analysis offers the first discussion of the tradeoffs inherent to these performance enhancing techniques in the context of SIP proxies.

- **Provide recommendations for optimal throughput and resource use by proxies:** By performing our experiments over a range of possible configurations, we are able to demonstrate that throughput is not maximized simply by using the maximum possible settings for parallelization and batching. Instead, carefully tuning settings can maximize throughput while using approximately 24% of the bandwidth required in naïve settings.

- **Characterize the delay added by batching to call setup time:** By running additional experiments, we demonstrate that the delay introduced by the use of batch requests is adequate even for time-sensitive applications such as VoIP. Our measurements show that this delay is at most 160 ms in our testbed and, therefore, does not cause unnecessary retransmissions (default retransmission time is 500 ms).

The remainder of this chapter is organized as follows: Section 4.1 offers an overview of the two main techniques used to improve Digest authentication performance in distributed scenarios; Section 4.2 describes our methodology and provides the details of our experimental testbed; Section 4.3 provides the results of our experimental evaluation for the different configurations studied; Section 4.4 presents a general discussion of the results obtained and additional considerations; Section 4.5 offers a summary of our work.

4.1 Improving SIP Authentication Performance in a Distributed Scenario

The SIP standard (e.g., RFC 3261 [161]) recommends several security mechanisms that provide user authentication (i.e., Digest authentication, SSL/TLS, IPsec, S/MIME). However, Digest authentication is the preferred mechanism in most SIP deployments because it is more efficient and easier to deploy than the other recommended mechanisms. SIP Digest authentication (see Section 2.2.3), a challenge-response authentication protocol, is more efficient because it relies on cryptographic hash operations (i.e., MD5 algorithm) which are computationally cheaper than using symmetric or public key cryptography.

In this work, we analyze the performance of Digest authentication in a SIP distributed scenario where multiple proxies share a central authentication service, similar to the nationwide SIP infrastructure scenario depicted in Figures 1 and 2 in Section 2.2.2. In this scenario, SIP proxies are located in areas geographically close to subscribers to reduce effects of network latency on session setup. The central authentication service can perform authentication operations itself (i.e., RADIUS, LDAP) or can store subscriber information required for authentication (i.e., database). In the latter case, the authentication service sends the subscriber's authentication credentials to the proxies, and the proxies perform the authentication operations. We followed this model in our study because it is the one typically used in Digest authentication.

While the use of a central database topology offers several advantages (i.e., simplified management and security), its performance is significantly reduced when Digest authentication is enabled (see Section 4.3). There are two main reasons for this performance problem: the larger network latency between the proxies and the database and the fact that Digest authentication requires a query to the database for each message authentication operation. Each time a proxy process authenticates a message,

it has to wait for the database response to continue processing the message. During this time, the proxy process can not serve new requests (blocking call). As a result, each proxy process handles a lower call throughput.

An often recommended technique to reduce the effects of network latency in distributed scenarios is *parallel execution*. The idea is to use multiple proxy processes per server. The higher the number of concurrent processes, the lower the probability that a client request will have to wait for a process to become available. As a result, the proxy can support higher call throughput. The trade-off, however, is higher hardware requirements such as CPU, memory and network bandwidth.

Another technique, that to the best of our knowledge has not been used before in SIP deployments, is *request batching*. The idea is to reduce the number of times the proxy needs to access the database so as to avoid the impact of the high round trip time between the proxies and the database. Instead of sending single queries to the database, the proxy holds requests in a queue of size n. Once the queue is full, the proxy sends a batch query to the database to retrieve the corresponding n authentication credentials in one single round trip. While this technique is more efficient than parallel execution (e.g., lower hardware requirements), our experimental analysis shows that it fails to achieve similar performance improvements. In addition, request batching adds delay to each requests in the batch. However, the delay introduced is tolerable even for VoIP applications (See Section 4.3.5).

As mentioned earlier, the use of multiple processes is an effective way to improve throughput. However, it is not efficient in terms of the bandwidth required between the proxy and the database. Request batching, alternatively, is not effective for maximizing throughput, but does improve performance with dramatic reductions in communications and application-layer overhead. The complementary nature of these two techniques makes their combination a logical next step. We propose and evaluate the use of a *hybrid approach*: combining multiple processes with batch requests.

Our experimental evaluation and analysis of this hybrid approach is presented in Section 4.3.4.

4.2 Experimental Setup

To characterize the performance of a SIP proxy with Digest authentication, we implemented an experimental test bed based on the nationwide VoIP provider scenario depicted in Figures 1 and 2 in Section 2.2.2. SIP proxies are located in areas geographically close to subscribers to reduce effects of network latency on session setup. The proxies share a central authentication service which stores subscriber information and performs authentication related operations.

The authentication service can perform the authentication operations itself (i.e., RADIUS, LDAP) or can simply store subscriber information. In the latter case, the authentication service sends the subscriber's authentication credentials to the proxies, and the proxies perform the authentication operations and then delete the credentials. This is the model used in SIP Digest authentication, and therefore, the one modeled in our study.

4.2.1 Testbed Configuration

Our experimental test bed consists of three main components:

- SIP Proxy: we used OpenSER 1.3.2 [147] (now known as OpenSIPS) as our SIP proxy. OpenSER is a mature and stable open source SIP proxy optimized for high performance. OpenSER was configured with minimal functionality (only required modules were enabled). A single stateless proxy was used in our tests. OpenSER was installed in a server with 8 2-GHz Quad-Core AMD Opteron processors, 16 GB of memory, 1 Gbit Ethernet card and running 2.6.24 Linux Kernel (Ubuntu 8.04.2).

- Remote database: MySQL 5.0.51a [140] was used as the database software for

storing OpenSER's configuration data and subscriber's information (including authentication credentials). A default configuration was used in our test (no database optimizations were used). The database was populated with 10,000 subscribers, all belonging to a single SIP domain. MySQL was installed in a server with the same hardware and operating system as the server used for the proxy.

- User Agents (UAs): to simulate the SIP workload generated by the UAs, we used SIPp 3.1 [75], an open source test tool and traffic generator for the SIP protocol. A total of 24 SIPp instances were used to generate the workload, divided in two groups: 12 UA clients (UAC) and 12 UA servers (UAS). The number of SIPp instances was determined based on the hardware resources available. A total of 7 servers were used for running the SIPp instances (multiple instances per server). The default SIPp scenarios were modified to support Digest authentication, according to the call dialog show in Figure 3 in Section 2.2.3 (only INVITE messages are authenticated). The SIPp servers had similar hardware and operating system as the servers used for the proxy and database.

The above components communicated using a dedicated Gigabit Ethernet network. To simulate the network latency between the proxy and the database, we used the Linux traffic control tool [28] with the network emulation module (netem). A network latency value of 30 ms was used in our experiments. This is a conservative value to a centralized location on the continent considering that the typical coast-to-coast network latency in the United States is between 80 to 100 ms.

Experiments were executed using a combination of Bash and Perl scripts. Additional open source tools such as iftop, pidstat, mpstat and vmstat were used to capture the required metrics in each test.

4.2.2 Adding Batch Requests Support to OpenSER

In our study, we evaluated the use of batch requests to improve proxy performance. The basic idea behind this approach is to avoid individual requests to the database. Instead, requests are temporarily stored at the proxy and sent together as one multi-condition request. For example, a batch request of size 2 uses the following SQL query syntax:

Select ha1 from subscriber where username='00033' OR username='002459'

The previous SQL query returns the authentication credentials of two subscribers using a single round trip to the database. Using this approach, we can reduce the impact of the bandwidth overhead associated with individual requests. More details about this technique are presented in Section 4.3.3.

OpenSER does not support batch requests. Therefore, it was necessary to modify OpenSER to add this mechanism. Our code was added to OpenSER's authentication module. However, it was also necessary to modify OpenSER's core components to enable batch requests. Our experimental code is available to the community as a software patch for OpenSER version 1.3.2 at `http://www.cc.gatech.edu/~idacosta/proxy_batch.html`

4.2.3 Methodology

Three configurations were evaluated in our tests: the use of multiple processes, the use of batch requests and the combination of both mechanisms (i.e., hybrid approach). To evaluate each configuration, we focused on the following metrics: call throughput at the proxy, message retransmissions, failed calls, bandwidth between the proxy and the database and CPU utilization for the proxy and database.

Call throughput corresponds to the number of successful calls per second (cps) measured in each time period (5 s). The maximum call throughput was determined as

the highest load where the number of failed calls remained under 1% of the total load (maximum usable throughput with 1% failure tolerance). We selected this bound as it is commonly applied in more traditional telephony networks. Message retransmission refers to the number of SIP messages being retransmitted due to the expiration of timers in the SIPp instances. We used the default retransmission time recommended by the SIP RFC: 500 ms. Failed calls correspond to the number of unsuccessful calls in the last period measured by the SIPp UASs. The two main reasons for call failures in our tests were unexpected messages (messages out of order) and maximum number of retransmissions (maximum number of UDP retransmission attempts has been reached). We used the default values in SIPp for UDP retransmissions: 5 for INVITE transactions and 7 for others.

Each test was run for 10 minutes. During this period, the SIPp instances generated an increasing SIP workload. The workload was increased every 5 seconds by a constant amount. The amount of increase was adjusted according to the configuration evaluated (depending on the number of processes running on the proxy), ranging from 12 to 180 cps (to avoid saturation of the proxy too quickly). During each test, performance statistics were collected by the UASs. Additional data (i.e., bandwidth and CPU utilization) was collected using the tools described in Section 4.2.1.

To ensure the validity of our results, each test was repeated 10 times. Average values were used in our analysis and a 95 % confidence interval is provided. Approximately 600 unique tests were executed during our study.

In the following section, we present the results of our tests for each configuration and our analysis.

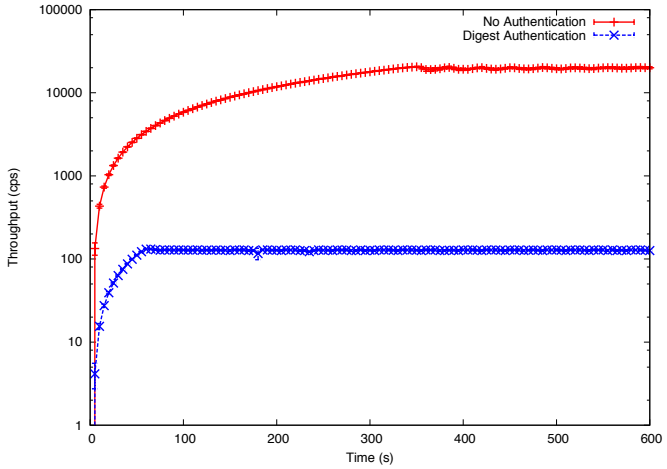


Figure 8: Proxy's throughput with and without Digest authentication. The network latency (30 ms) between the proxy and the database significantly reduces proxy's performance

4.3 Analysis of Throughput Enhancement Techniques

4.3.1 Standard Configuration

Our first test evaluated the performance of two stateless SIP proxy configurations: non-authentication (base case) and Digest authentication with remote database (RTT=30 ms). Both configurations used 4 child processes (default OpenSER value). The results are presented in Figure 8.

For the non-authentication configuration, the maximum throughput registered was approximately 20,000 cps. However, this was not the maximum throughput that the proxy could handle. At 20,000 cps the SIPp instances were unable to continue increasing the workload. In other words, the SIPp instances reached saturation before the proxy. At 20,000 cps the proxy had a CPU utilization of 20%. Therefore, we can estimate that the maximum throughput that our proxy can handle is significantly larger.

For the Digest authentication configuration, the maximum throughput registered was approximately 130 cps. This value represents a drop in performance of almost three orders of magnitude, but is logically correct as it closely matches the estimation

$1000\ ms/30\ ms\ delay = 33 * 4\ processes$. This drop in performance also means that the proxy and database are heavily underutilized (less than 1% CPU utilization).

The main reason for this significant drop in performance is the network latency between the proxy and the database. When Digest authentication is not used, the proxy routes messages very quickly (less than 50 μs per message). However, when Digest authentication is enabled, the proxy needs to contact the database to get the subscriber's authentication credentials to verify the subscriber's request (AUTH box in Figure 3 in Section 2.2.3). This operation adds approximately 30 ms to the total session setup. This additional time slows down the processing of messages in the proxy, because each OpenSER process routes messages serially. Each time an OpenSER process needs to contact the database, it stops processing other messages and waits for the database's response (blocking call).

A popular approach to reduce the effect of blocking calls and network latency is to increase the number of parallel operations performed by the proxy. In other words, increase the number of OpenSER child processes. This approach is often recommended by OpenSER developers to improve performance. In the next section, we present the evaluation of this option.

4.3.2 Improving Performance with Multiple Processes

To evaluate the use of multiple processes to improve performance, we measured the proxy's throughput over a range: from 2 to 1024 processes, increasing our interval by a factor of 2. Figure 9 shows the results of our experiments. As expected, the proxy's throughput improves as the number of processes increases. Intuitively, this improvement is due to the increased probability that a non-blocked process will be available when a new request arrives at the proxy.

It is important to notice that the throughput values in Figure 9 represent the *maximum usable throughput* (1% call failure tolerance) measured in our tests. In

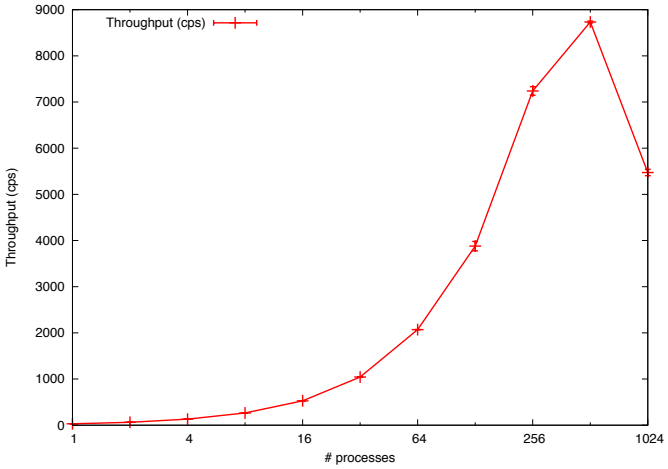


Figure 9: Proxy's maximum usable throughput (<1% failed calls) for different number of processes. Proxy's performance improves with higher number of processes. However, performance drops sharply when 1024 processes are used.

most cases, this value is equivalent to the maximum throughput handled by the proxy. However, in certain configurations, the maximum throughput has a high percentage of call failures (>> 1%), which makes this value impractical in production environments.

Figure 9 depicts the maximum usable throughput for 512 processes: almost 9,000 cps. This value represents a considerable improvement in performance when compared with the value obtained in the previous section (130 cps). Our results also show that at 1024 processes, the maximum usable throughput decreased markedly to approximately 5,500 cps. Even though the performance improvement is significant, it still falls short when compared to the proxy's throughput with no authentication (approximately 20,000 cps). To understand why the proxy can not handle higher throughput, we analyzed in more detail the differences between configurations using 512 and 1024 processes.

Figure 10 provides a more in-depth characterization of the behavior of 512 processes. As previously mentioned, traffic is gradually increased with time to create performance profiles. Two critical observations can be made from this configuration. First, the maximum throughput is approximately 13,000 cps; however, the corresponding rate of failed calls is intolerably high (almost 21% of the total throughput).

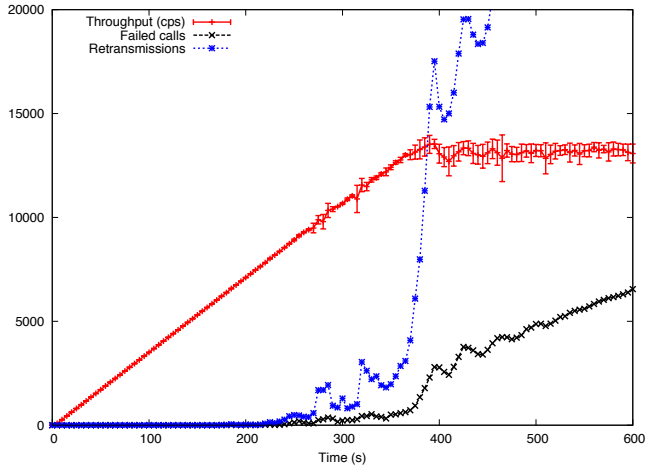


Figure 10: Proxy's throughput for the 512 processes configuration. The maximum throughput that the proxy can handle is limited by the database maximum throughput.

Second, message retransmissions begin before reaching the maximum throughput and grow extremely rapidly once it is reached. The number of failed calls also grows faster once the maximum throughput is reached. Typically, the number of retransmissions and failed calls increases because the proxy can not process the messages fast enough (the proxy is too busy). However, the proxy's CPU utilization at the maximum throughput was approximately 40%. Figure 12 provides additional context to clarify this discrepancy. Specifically, around 512 processes the database application is using more than 100 % CPU utilization (multiprocessor machine). At 512 processes, the database application is using almost 120% CPU utilization. At this point the database can not process a higher rate of requests from the proxy, and as a result, the database becomes the performance bottleneck.

Based on our results and assuming the use of a faster database (i.e., an in-memory database), we could continue increasing the number of processes to improve the proxy's throughput. For example, the proxy could theoretically handle a throughput of almost 35,000 cps with a 100% CPU utilization (13,000 cps uses 40% CPU utilization). At that point, the proxy's CPU will be the bottleneck. However, Figure 11 contradicts this assertion. Note that retransmissions and call failures happen earlier

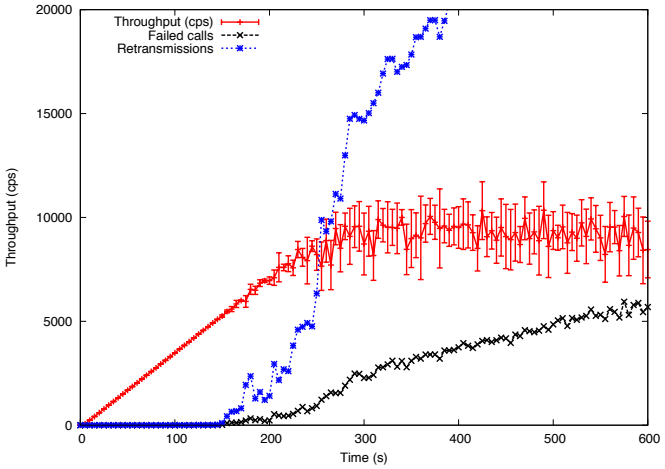


Figure 11: Proxy's throughput for 1024 processes configuration. Trashing in the proxy due to the high number of processes degrades proxy's performance

(at a lower traffic rate) during the tests on 1024 process when compared with the 512 processes case. The maximum throughput for 1024 processes is also less than the 512 process case, indicating that the database is not the only cause for the drop in performance in this configuration. The reason for this behavior is that the high number of processes begins to degrade the proxy's performance due to thrashing (an elevated number of page faults and context switches). As a result, we can conclude that only increasing the number of processes is not enough to maximize throughput.

The use of simple parallelization techniques also causes bandwidth usage between the proxy and the database to become a concern. Figure 13 shows the bandwidth for the queries-to and responses-from the database. The response bandwidth is higher than the query bandwidth because the queries have smaller size than the corresponding responses (authentication credentials). Note that bandwidth grows linearly with the number of processes. For example, 512 processes require a bandwidth of 12 Mbps for queries and 24 Mbps for responses. These bandwidth values can be prohibitive for some scenarios, specially if the database and the proxy communicate across the public Internet. The bandwidth between the proxy and the database may therefore limit the number of processes that can be used at the proxy.

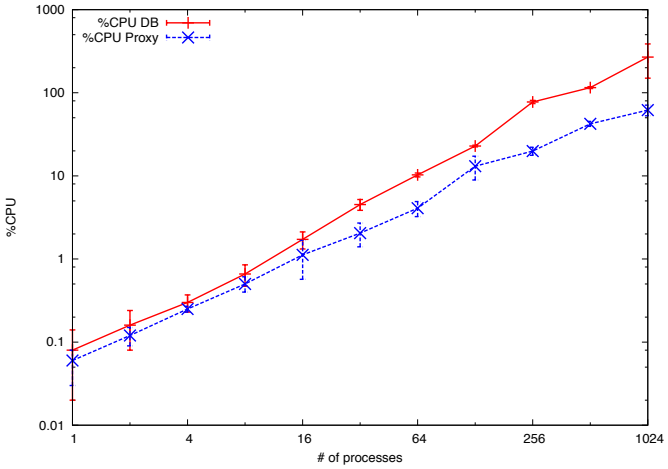


Figure 12: Proxy and database CPU utilization for different number of processes. Around 512, the database CPU utilization is more than 100 % and the database becomes a bottleneck for proxy's performance

The network and application layer overheads are the main reason for the high bandwidth requirements. A single query has a packet length of 126 bytes, with 56 bytes of payload (56% overhead). The corresponding response has a packet length of 247 bytes with 146 bytes of payload (41% overhead). Therefore, almost half of the traffic exchanged between the proxy and the database corresponds to network and application layer headers and control packets. The situation is made worse if we consider the use of security protocols such as IPsec or SSL to protect this communication channel.

Finally, we note that OpenSER creates an independent TCP connection to the database for each child process it spawns. This approach is not very efficient if a high number of processes are used (i.e., a TCP port per process needs to be allocated). A more efficient approach could be to share a pool of TCP connections to the database among all the processes. However, we did not attempt to retrofit this codebase to include this optimization.

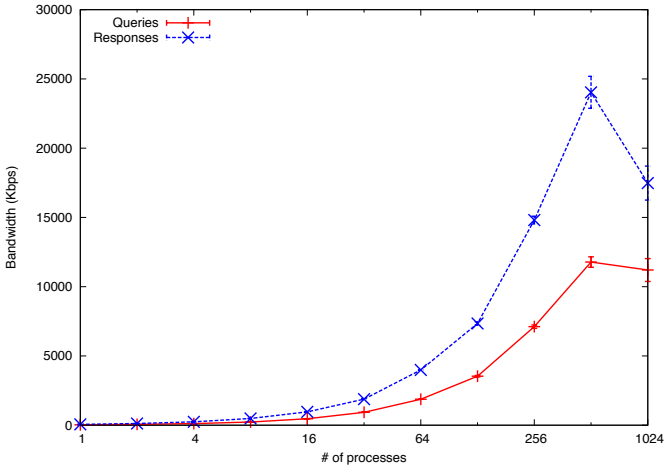


Figure 13: Bandwidth between the proxy and the database for different number of processes. A considerable amount of bandwidth is required for high number of processes

4.3.3 Improving Performance with Batch Requests

The second approach to improve proxy performance is the use of request batching. The idea is to reduce the number of times the proxy needs to access the database so as to avoid the impact of the high round trip time between the proxy and the database (the most expensive step of the authentication process). Instead of sending a single query to the database, the proxy holds requests in a queue of size n. Once the queue is full,[1] the proxy sends a batch query to the database to retrieve the corresponding n authentication credentials in one single round trip.

Based on the above, the time a batch query takes to retrieve n credentials should be less than the total time n single queries take to retrieve n credentials, as the following inequality shows:

$$RTT_{pd} + t_{batch} < n \times (t_{db} + RTT_{pd}) \quad (1)$$

[1]Because we have relatively high amounts of traffic, we are not worried about starving requests by forcing them to wait for the queue to fill. A real deployment could avoid such an issue by incorporating a timer, which would cause the batch to be sent even if the queue was not full after some period.

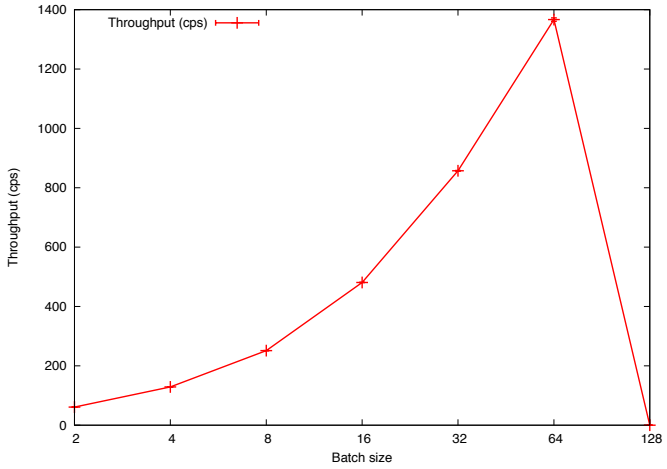


Figure 14: Proxy's maximum usable throughput for different batch sizes using a single process. The performance improvement is lower when compared with multiple processes.

where RTT_{pd} is the network latency between the proxy and the database, t_{batch} is the query execution time of a batch query of size n, and t_{db} is the query execution time of a single query (query execution time corresponds to the time the database takes to execute a query).

Assuming that $t_{batch} = n \times t_{db}$ (worst case scenario), the impact of network latency is reduced by a factor n when a batch query is used instead of single queries. Using this result, we can estimate the time t_{sq} each query in the batch takes:

$$t_{sq} = \frac{RTT_{pd} + t_{batch}}{n} \tag{2}$$

Assuming $t_{batch} = n \times t_{db}$ we have:

$$t_{sq} = \frac{RTT_{pd}}{n} + t_{db} \tag{3}$$

Equation 3 shows that the impact of the network latency is effectively amortized among the n queries in the batch.

To evaluate the effectiveness of using batch requests, we measured the proxy's maximum usable throughput for different batch sizes: from 2 to 128, increasing the

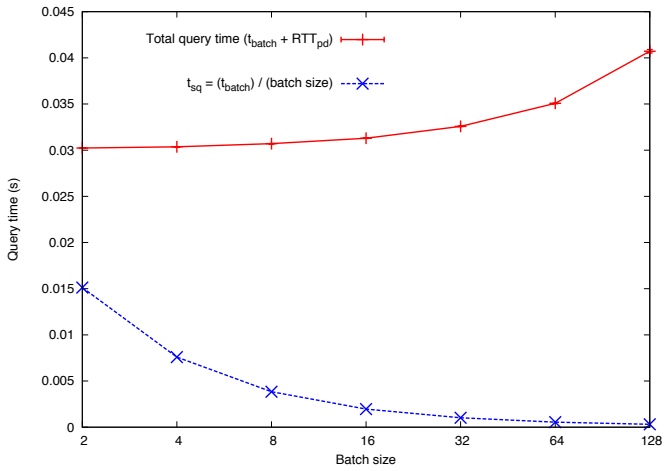


Figure 15: Total query time ($t_{batch} + RTT_{pd}$) and $t_{sq} = \frac{RTT_{pd}+t_{batch}}{n}$ for different batch sizes. Using batch sizes larger than 64 has little effect on improving performance

batch size by a factor of 2 in each step. Figure 14 shows the results of these tests. The maximum usable throughput occurred with a batch size of 64: almost 1,400 cps. This performance improvement is small when compared with the parallel execution approach, but note that these tests where executed with a single process. It can therefore already be concluded that batch requests alone are not enough to maximize proxy's performance.

Figure 14 also shows that the maximum usable throughput with a batch size of 128 is close to 0 cps. With this batch size, the number of retransmissions and failed calls is extremely high almost immediately after the start of the experiments. These results provide us with an estimation of the maximum batch size that can be used in our scenario.

Due to the lower throughput values obtained using only batch requests, the CPU utilization in both, the proxy and the database, was less than 10 % for all the tests.

To confirm the validity of Equation 1, we measured the total query time ($t_{batch} + RTT_{pd}$) for different batch sizes. Figure 15 depicts how the batch query time increases with the batch size. However the rate of increase is much lower than using single queries. For example, a single query takes approximately 30.2 ms, therefore, 32

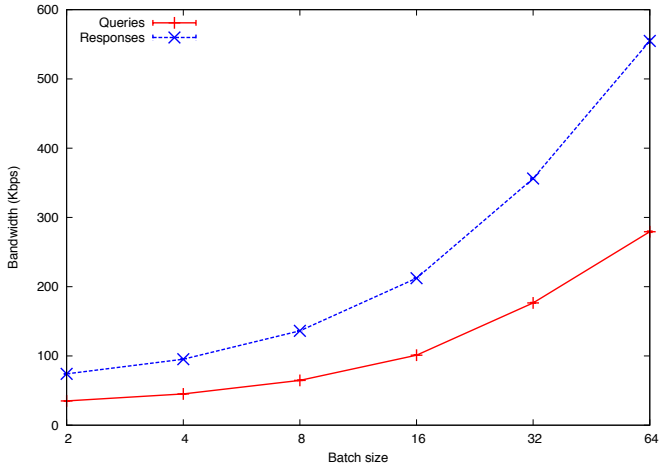


Figure 16: Bandwidth between the proxy and the database for different batch sizes. Batch requests have lower bandwidth requirements than multiple processes

queries will take approximately 964.6 ms. Instead, a batch query of size 32 takes approximately 32.581 ms. Figure 15 also shows how the network latency is amortized among the queries in the batch (Equation 3). It is important to notice that after a batch size of 64, the curve begins to flatten out. This behavior indicates that larger batch sizes will have little effect on reducing the impact of network latency.

Figure 16 shows that the use of batch requests also reduces the bandwidth requirements between the proxy and the database. For example, a proxy with 8 processes has a maximum throughput of 265 cps and requires 232 and 482 Kbps for query and response bandwidth respectively. The same proxy with a single process and a batch size of 8 has a maximum throughput of 251 cps and requires 64.70 and 136.20 Kbps for query and response traffic respectively. Batching therefore required 71.8% less bandwidth than recommended multiple processes technique for the same throughput.

Batch requests require lower bandwidth because they have better payload efficiency (payload/total packet size) than multiple processes. Figure 17 shows how batch requests have a lower total packet length than the similar number of single queries. For example, 8 single queries will use a total packet length of 1,008 and 1,976 bytes for the query and the response, respectively. A batch query of size 8, will

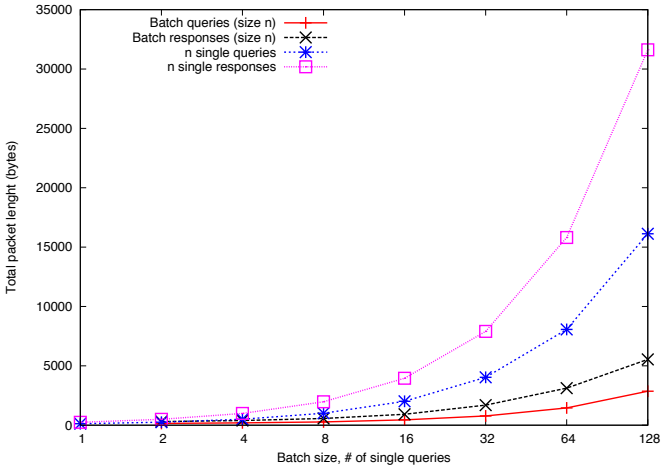


Figure 17: Total packet length for batch queries and single queries. Batch queries have better payload efficiency than the equivalent number of single queries

require 277 and 569 bytes instead. A batch query of size 8 has a payload of 207 bytes and 450 bytes for the query and the response packets (25% and 21% overhead); 8 single queries have a payload of 448 and 1,168 bytes (56% and 41% overhead).

A tradeoff with the use of batch requests is added latency in session establishment time associated with the queuing period. In particular, the first request to arrive to the queue will have the longest delay (worst case). This request will have to wait for the queue to fill up first in order to continue with authentication procedures. However, this additional latency is virtually unnoticeable to users as session setups in telephony are on the order of seconds [67].

Given all of these inputs, we now attempt to select an appropriate batch size. In general, this will be a scenario dependent value. Equation 3 indicates that using larger batch sizes is the best strategy to reduce the effect of network latency. However, increasing the batch size will also increase the execution time on the database (t_{db}), as shown in Figure 15. After a certain point, increasing the batch size will no longer improve performance, especially if the database is not optimized for processing large batch requests. Our results show that for our scenario, batch sizes larger than 64 will not help to improve performance; on the contrary, they will degrade it (see Figure 14).

There are also other factors to consider when choosing the batch size. Using larger batch sizes will also increase the delay introduced to the session setup, and produce unnecessary retransmissions and failed calls if the queue is not filled up fast enough. In general, the queue has to be filled up before the retransmission time of the first request in the queue expires (default retransmission time is 500 ms). Using larger batch sizes will also increase the length of the queries and responses packets. Once the packet length is bigger than the network MTU (Maximum Transmission Unit), fragmentation will occur. Fragmentation decreases the payload efficiency because additional control information is required (i.e., more headers). Even worse, fragmentation will also increase the likelihood of packet loss, and therefore, excessive retransmissions (retransmission of all the fragments). This factor is especially important in Internet scenarios. In Figure 17, a batch request of size 128 requires 2 fragments for the queries, and 4 fragments for the responses (plus additional TCP control packets).

A final factor to consider is the proxy's design and implementation. We found several problems and error messages with batch sizes of 128 or larger. We concluded that OpenSER's design is one of the main reasons for the performance degradation when a batch size of 128 was used.

4.3.4 Hybrid Approach: Combining Multiple Processes with Batch Requests

The use of multiple processes is an effective way to improve throughput. However, it is not efficient in terms of the bandwidth used between the proxy and the database. Request batching, alternatively, is not effective for maximizing throughput, but does improve performance with dramatic reductions in communications and application-layer overhead. The complimentary nature of these two techniques makes their combination a logical next step. To verify that a hybrid of these two mechanisms can

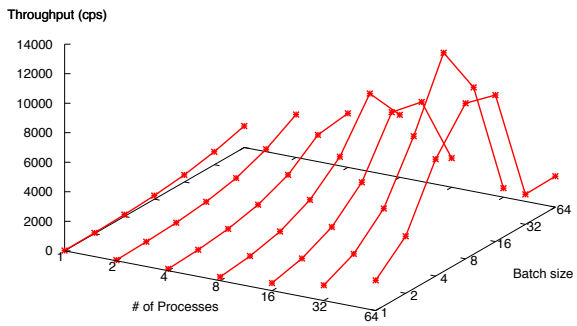


Figure 18: Proxy's maximum usable throughput for different combinations of number of processes and batch sizes. The best performance is achieved with the 32 processes with batch size of 16

further improve performance, we tested the combination of different number of processes (2, 4, 8, 16, 32, and 64) with different batch sizes (2, 4, 8, 16, 32, and 64) - a total of 36 tests.

Figure 18 shows the maximum usable throughput values for each configuration. The maximum usable throughput is reached with the combination of 32 processes with a batch size of 16 (32p-16b): approximately 12,100 cps. This value represents a 34% improvement over the maximum usable throughput obtained with multiple processes (9,000 cps for 512 processes).

Figure 18 also shows that configurations with higher number of processes or larger batch sizes than 32p-16b (i.e., 32p-32p or 64p-32p) exhibit poorer performance than the 32p-16b configuration. These other configurations are able to handle approximately the same maximum throughput than 32p-16b but also have more retransmissions and call failures, which degrades their usable throughput. Such degradation is caused by a combination of database saturation and the operation issues associated with OpenSER and SIPp. If a faster database were used (e.g., in-memory), these configurations would likely perform better than 32p-16b; however, our characterization is valuable as it describes how OpenSER is currently configured.

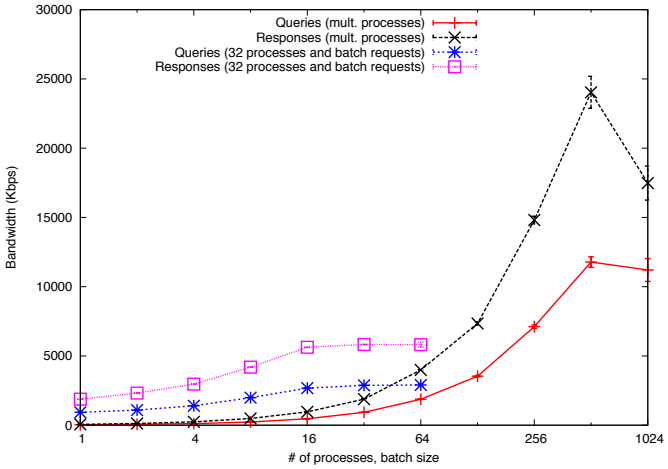


Figure 19: Bandwidth between the proxy and the database for different number of processes and for 32 processes with different batch sizes. The combination of multiple processes with batching required significantly less bandwidth than using multiple processes only.

Configurations such as 16p-32b also perform worse than the 32p-16b scenario. While these configurations appear to be equivalent, in practice they are not. The reason is that the use of a higher number of processes is more effective to maximize throughput than using higher batch sizes.

Figure 19 presents the measured bandwidth between the proxy and the database for parallel execution and for the 32 processes hybrid approach. The figure shows a significant difference between the bandwidth used by multiple processes and the bandwidth used by the hybrid approach. For example, the best hybrid configuration tested (32p-16b) requires 2,676 Kbps for queries to the database and 5,625 Kbps for responses from the database. In contrast, the best multiple processes configuration (512p) requires 11,780 Kbps for queries to the database and 24,030 Kbps for the responses from the database. This is an improvement of 77.3% and 76.6% for requests and responses, respectively. Therefore, the 32p-16b configuration requires 4 times less bandwidth than the 512p configuration. The main reason for this significant difference is that batch requests have better payload efficiency than single queries (see Section 4.3.3).

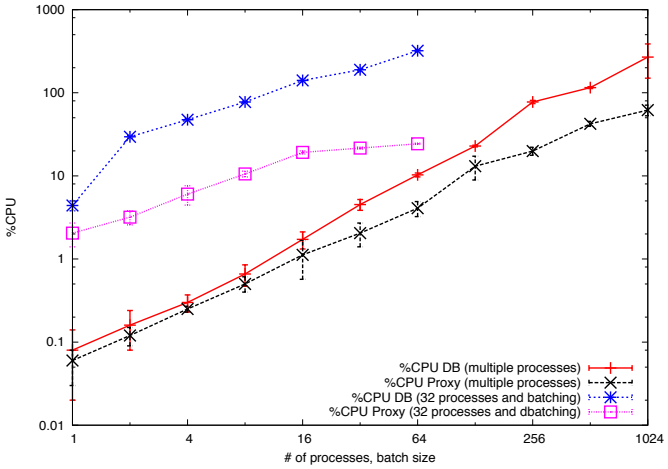


Figure 20: Proxy and database CPU utilization for different number of processes and for 32 processes with different batch sizes. The combination of multiple processes with batching requires less proxy CPU time than multiple processes

Finally, we compare the CPU utilization in the proxy and the database for the hybrid approach. Figure 20 compares the CPU utilization for different number of processes against the CPU utilization for 32 processes with different batch sizes. In the database's case, the figure shows that there is not a significant difference between using multiple processes or the hybrid approach; both techniques saturate the database (more than 100% CPU utilization). However, in the proxy's case, the hybrid approach uses less CPU time than multiple processes. For example, for the 32p-16 case, the CPU utilization is almost 20%. For 512 processes, the proxy CPU utilization is almost 42%. Figure 21 depicts this difference in greater detail.

To understand why the 512p configuration requires higher proxy CPU utilization, we compare Figure 10 and Figure 22. Both configurations handled approximately the same maximum throughput, however, the 512p configuration has more retransmissions and call failures earlier than the 32p-16b configuration. As a result, the proxy has to do extra processing when the 512p configuration is used. In addition, the use of a high number of processes increases the probability of more overhead (i.e., more context switching) in the proxy, consuming additional CPU time.

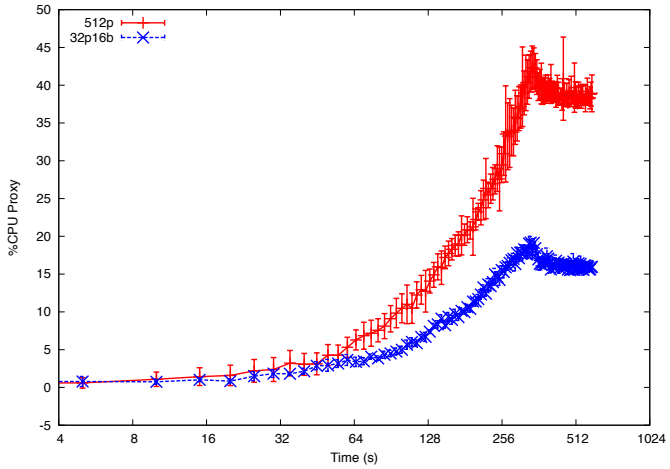


Figure 21: Proxy CPU utilization for 512 processes and for 32 processes with batch size of 16. Both configurations handle approximately the same maximum throughput but the hybrid approach requires less CPU time

4.3.5 Analysis of the Delay Introduced by Batch Requests

The use of batch requests adds a delay to most of the messages that require authentication. The reason is that queries to the database need to wait for the queue to fill up in order to be sent to the database. In general, the longest delay is experienced by the first query to arrive to the last queue that fills up (each process has an independent queue). The longest delay can be estimated as follows:

$$t_{dmax} = \frac{n * m}{r} \tag{4}$$

where n is the size of the queue, m is the number of child processes in the proxy and r is the expected call throughput (calls per second). For example, for 32p-16b hybrid configuration and a call throughput of 2,000 cps, the longest delay added by batch requests is approximately 256 ms.

To avoid unnecessary retransmissions, the delay added by batch requests should be sufficiently small to avoid the expiration of the retransmission timer in the UACs.

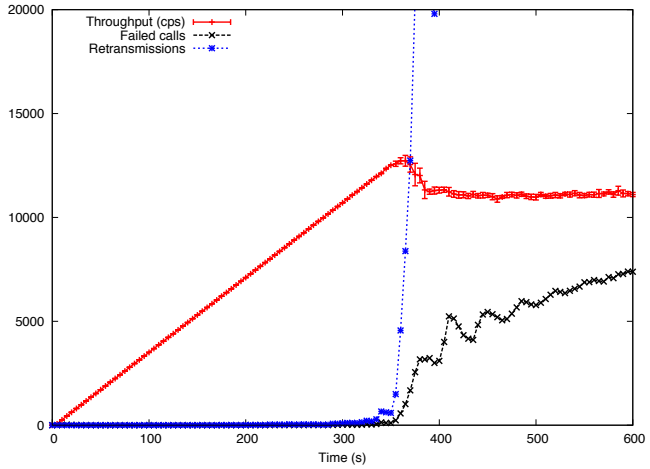


Figure 22: Throughput for 32 processes with batch size of 16. Notice the lower number of retransmissions and failed calls when compared with the 512 processes configuration (Figure 10)

This constraint is expressed in the following inequality:

$$RTT_{cp} + t_{auth} + t_{dmax} < t_{ret} \tag{5}$$

where RTT_{cp} is the round-trip time between the clients (UAC and UAS) and the proxy, t_{auth} is the time required by a non-batching authentication operation (including the query to the database), t_{dmax} is the maximum delay added by the use of batch requests (equation 4) and t_{ret} is the value of the retransmission timer in the UAs (e.g., 500 ms). This inequality assumes that the UAS answers the call request immediately.

The additional delay introduced by batch requests increases the setup time of each call. Therefore, we ran an experiment to evaluate how the call setup time is affected by batch requests. In this experiment, a proxy with a 32p-16b hybrid configuration received a constant call load of 6,000 cps during 20 seconds. In this period, we measured the setup time for all the calls established (120,000 calls). The results are shown in the histogram presented in Figure 23 (25 ms bins). As this Figure depicts, 98 % of the calls were established in less than 200 ms (99.90 % in less than 350 ms). Using equation 5 and taking into account that in our testbed $RTT_{cp} \approx 10$ ms and

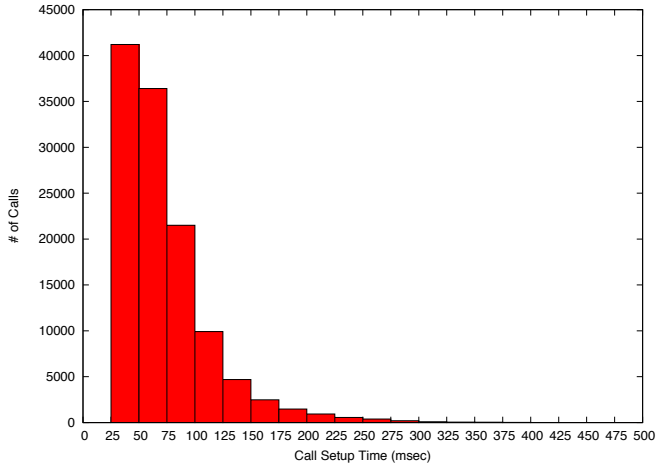


Figure 23: Call setup time distribution for 120,000 calls measured using a constant call throughput (6,000 cps) and a 32p-16b proxy configuration. Most of the calls (99.90 %) were established in less than 350 ms.

$t_{auth} \approx 30$ ms (RTT_{pd}), the delay added by batch requests is at most 160 ms for nearly all the calls. This delay is small when compared to the default retransmission time (500 ms) and unlikely to be noticed by humans. As a result, we can conclude that the delay added by batch requests is acceptable for VoIP applications.

As equations 4 and 5 show, the delay introduced by batch requests is proportional to the inverse of the call throughput. Therefore, call throughput below certain threshold (equation 4) will cause retransmissions. One option to avoid retransmissions is to implement a timer that, on expiration, will indicate the proxy to send the batch request to the database even if the queue is not full. Other option is to monitor the call throughput and adjust the size of the queue dynamically to avoid unnecessary retransmissions.

4.3.6 Evaluating Performance with Multiple Proxies

The testbed used in all previous experiments consisted of servers with high-end hardware specifications, capable of supporting carrier level SIP traffic. However, only one proxy and one database were used in our experiments due to resource constraints and the high capacity of the proxy and database servers (e.g., 8 servers were required to

Table 1: Performance results for multiple proxies including 95% confidence intervals

Configuration	Max. Usable Throughput (cps)	DB CPU (%)	DB send/recv BW (KBps)
128p	1254.17 (±125.86)	114.20 (±1.70)	538.43 (±45.47) / 207.54 (±17.48)
32p16b	1379.26 (±141.32)	65.70 (±4.9)	128.31 (±18.13) / 52.71 (±6.25)

generate enough load to reach the limits of the system). Therefore, a new testbed based on the Georgia Tech Emulab network testbed was implemented to evaluate an scenario with multiple proxies. The new testbed consisted of 3 proxies and one database and different network latencies between each proxy and the database (30, 45 and 70 ms). However, the server nodes in Emulab have much lower capacity than the servers used in previous experiments. For example, Emulab nodes use virtual machine technology (guest OS), have lower number of processors (e.g., dual-core machines), and significantly less memory (512 MB).

In spite of its lower capacity, the new testbed allowed us to evaluate two configurations: multiple processes and the hybrid approach. For multiple processes, a configuration of 128 child processes per proxy was evaluated (larger number of processes caused instability problems). For the hybrid approach, we used a 32p-16b configuration (best hybrid configuration). The results are presented in Table 1. The first thing to notice is the lower performance values supported by the Emulab testbed: less than 1,500 cps. However, despite of the lower call throughput supported by the testbed, the new results coincide with our earlier findings. We can see that the maximum usable call throughput supported by both configurations was approximately the same. However, the hybrid configuration required less bandwidth and database CPU utilization than the multiple processes approach (the hybrid approach also required less CPU utilization in the proxies: approximately 5 % less). These results confirm that the hybrid approach is more efficient that using only multiple processes to maximize performance in a multiple-proxy configuration.

Table 2: Best configuration for each query technique: multiple processes, batch requests and hybrid approach.

Configuration	Throughput (cps)	Send/Recv BW (Kbps)	Proxy CPU (%)	DB CPU (%)
512p	9,000	11,780 / 24,030	42	115
64b	1,366	279 / 555	6	2
32p16b	12,100	2,676 / 5,625	20	135

4.4 Discussion

Table 2 shows the results of the best configuration of each technique: multiple processes (512 processes), batch requests (one process with batch size of 64), and the hybrid approach (32 processes with bath size of 16). These results demonstrate that the hybrid approach effectively combines the good properties of the other two approaches: improved throughput values and lower bandwidth between the proxy and the database.

Based on the above results, 32p-16b provides the best performance for a distributed proxy architecture with similar components. We note, however, the need to carefully interpret these numbers. While the results indicate that the hybrid approach can handle higher throughput than the multiple processes approach, we did not test every possible configuration for each technique. It may be possible that a different number of multiple processes (somewhere between 256 and 512) offers performance closer to that of the 32p-16b configuration. Such a configuration is still unlikely to improve over the hybrid approach due to the associated bandwidth overhead. It may also be possible that a different mix of multiple processes and batch size have even better performance. Accordingly, these results should be used as a guide and aid network administrators in their parameter setting. Both the 512p and 32p-16b configurations can handle approximately the same maximum throughput ($\approx$ 12,000 cps), which is bounded by the maximum database's throughput. The difference resides in the number of retransmissions and call failures that happen in each configuration, which results in different maximum usable throughput values. This difference can

be observed comparing Figure 10 and Figure 22. As a result, *the hybrid approach can maximize throughput as effectively as running multiple processes but using lower number of processes.* The use of lower number of processes helps to reduce context switching overheads and thrashing situations.

In terms of bandwidth requirements, the hybrid approach is much more efficient than any configuration of multiple processes. As Figure 13 shows, configurations larger than 128p require considerable bandwidth between the proxy and the database. In conclusion, *for similar throughput values, a hybrid configuration will require much less bandwidth than a multiple processes configuration.*

CPU utilization in the database is approximately the same for both, the hybrid and multiple processes techniques. However, the CPU utilization in the proxy will be higher for multiple processes because of the context switching overhead. Therefore, *for comparable throughput values, the hybrid approach will require the same or less proxy CPU time than a multiple processes configuration*

As a tradeoff, the use of batch requests will increase the session setup time in low traffic scenarios. However, the delay introduced is unlikely to be noticed by humans. Additionally, request batching increases the packet length of the queries and responses between the proxy and the database. Larger packet sizes may require fragmentation, making this approach more susceptible to packet losses at very large batch sizes.

In general, designers and developers can use the following rule when applying the hybrid approach: *for comparable throughput values, increasing the number of processes improves throughput to a point, but the use of request batching will be required to reduce the overhead associated with this technique in order to truly maximize the usable throughput of the proxy.*

It is not a coincidence that the to number of processes in the multiple processes approach is equal to the product of the number of processes and its batch size in the hybrid approach (512=32*16). If we look again at Figures 9 and 14, we can see that

for batch sizes lower than 16, approximately the same throughput is handled using n multiple processes, or using one process with a batch size of n. Therefore, *a configuration using n multiple processes will handle approximately the same throughput as a configuration with p processes with batch size=q, where p and q are factors of n and* $q \leq 16$.

When using the hybrid approach in a production environment, we recommend having batch requests enabled all the time. As mentioned before, the extra-delay introduced by batch requests is unlikely to be noticed by humans. Also, the use of timers will avoid unnecessary retransmissions due to timeouts when the call load is low. In this way, the system will be always prepared for events when the call load increases unexpectedly (i.e., flash crowds, distributed denial of service attacks and emergency situations).

Finally, while production environments may have different topologies and configurations, our testbed can not be considered unrealistic. For example, the use of multiple proxies sharing a single database is an option suggested by VoIP software providers [149]. In addition, our testbed is also based in a common topology used in cellular networks: a central database (the Home Location Register - HLR) servicing multiple proxy servers (the Mobile Switching Center/Visitor Location Register - MSC/VLR) closer to the clients. Therefore, our findings are relevant for production environments with distributed SIP topologies.

4.5 Summary

SIP proxies are often distributed across a wide geographic area in order to minimize latency between themselves and clients. However, the supporting authentication services are often centrally located, potentially leading to the degradation of call throughput due to network latency. In this chapter we have analyzed two schemes to improve proxy throughput in such an architecture. First, we demonstrated that

the commonly recommended approach, parallel execution, improved performance but quickly caused the rate of call failure to rise. We then implemented a request batching mechanism that reduced the bandwidth overhead associated with previous approach, but failed to reach a sufficiently high throughput. Second, we created a hybrid of these two schemes that improved throughput by more than 34%, reduced bandwidth overhead by more than 75% over the best parallel execution method and maintained loss rates below 1%. Finally, we also characterized the delay added by batch requests to the setup time of each call. Our results showed that this delay is acceptable even for time-sensitive applications such as VoIP.

CHAPTER V

PROXYCHAIN: DEVELOPING A ROBUST AND EFFICIENT AUTHENTICATION INFRASTRUCTURE FOR CARRIER-SCALE VOIP NETWORKS

Voice over IP (VoIP) is fundamentally reshaping the telephony landscape. Instead of using dedicated, circuit-switched lines, VoIP allows for phone calls to be multiplexed with other data traffic over the Internet. This convergence between voice and data communications provides a number of benefits. For instance, providers can now offer a range of new services such as video and presence. Unfortunately, the transition from traditional phone networks to VoIP also creates a number of new security challenges.

Authentication represents one of the most important security issues facing VoIP systems. Providers have responded by implementing a number of security mechanisms, ranging from SSL/TLS to Digest authentication (see Section 3.1.2). Unfortunately, none of the suggested schemes are simultaneously strong, efficient *and* scalable enough to meet the needs of carrier-scale networks without vastly increasing the amount of deployed infrastructure.

SIP Digest authentication (see Section 2.2.3) is still the preferred authentication mechanism due to is simplicity and efficiency. However, despite the performance benefits achieved by our hybrid request approach in the previous chapter, the throughput and bandwidth overheads introduced by Digest authentication can still be too high for a large-scale VoIP application. These overheads are the main reason why not all the SIP message requests are authenticated during a session, which allows a variety attacks [207, 3]. Moreover, Digest authentication has several known weaknesses such as vulnerability to offline dictionary attacks and lack of mutual authentication. Thus,

a more robust and efficient SIP authentication mechanisms is required for large-scale VoIP applications.

In this chapter, we describe *Proxychain*, a robust and efficient authentication infrastructure designed to support operations in carrier-scale VoIP networks. Our solution is built around a single centralized authentication service working with proxy nodes distributed across a wide geographic area. We reduce the impact of the latency and load associated with this architecture by using a modified hash chain construction (a sequence of one-time authentication tokens generated by applying a hash function repeatedly, once-per token, to a secret root value). In addition to providing an efficient mechanism for mutual authentication, our approach also provides improved scalability through the secure caching of temporary authentication tokens at the proxies. To the best of our knowledge, Proxychain is the first protocol that applies the concept of hash chains in the SIP domain. Proxychain not only adapts this idea to SIP authentication but also extends it by including additional modifications that solve some of the weaknesses associated with hash chain protocols, resulting in a more robust protocol.

Moreover, improvements to the efficiency of SIP authentication afforded by Proxychain allow us to significantly increase the overall security of VoIP systems. For instance, several recently disclosed attacks on VoIP systems [207, 3] can be mitigated by simply having an authentication infrastructure scalable enough to cryptographically verify the origin of multiple SIP signaling request types (e.g., INVITE and BYE).

This work presents the following key contributions:

- **Design and implementation of a robust and efficient SIP authentication protocol for large-scale VoIP applications:** We develop Proxychain, a protocol based on modified hash chains. Our construction not only dramatically reduces the load on the centralized authentication database and the latencies

associated with accessing it, but also provides mutual authentication for clients and providers.

- **Evaluation of Proxychain through an extensive measurement study:** We measure, characterize and compare the performance characteristics of our proposed infrastructure against commonly used mechanisms. Our results show up to a 1,700% improvement over such schemes. Moreover, we demonstrate the ability to support the authentication needs of a national-scale VoIP network using unoptimized COTS hardware and databases.

- **Evidence of robustness to outages and downtime:** We demonstrate that our construction allows the network to operate during planned and unplanned outages, and estimate its robustness to such incidents. We show the ability to support normal operations with high availability for approximately 6 hours using only 28 minutes of preemptive computation.

The remainder of this chapter is organized as follows: Section 5.1 discusses the different problems associated with the use of Digest authentication; Section 5.2 details our proposed protocol; Section 5.3 provides the details of our experimental setup and methodology; Section 5.4 presents the results of our experiments; Section 5.5 discusses a number of additional points; Section 5.6 offers a summary of our work.

5.1 Problems with Digest Authentication

While more efficient, Digest authentication is less secure than protocols such as TLS or IPsec. For instance, it does not provide mutual authentication and complete message integrity. Limited integrity protection is offered but it is optional and not widely supported by UAs. Additionally, current implementations actually send the shared secret from the database to the proxy in order to calculate the correct client response. This approach is dangerous if the proxy is compromised. Several vulnerabilities have

been published regarding commercial SIP deployments due to these weaknesses [207]. More robust alternatives have been proposed (see Section 3.1.2), but they have not been adopted due to their deployment and operational costs.

The use of Digest authentication in an environment with a remote authentication service can reduce performance significantly, as the previous chapter showed (see Section 4.3.1). The main reason is that authentication operations become more expensive – the round-trip time (RTT) between a proxy and the database (tens of milliseconds) is now added to each authentication operation (hundreds of microseconds). The additional time added per call setup reduces the call throughput of each proxy. The problem is exacerbated by the fact that proxies have to query the database for each SIP message that requires authentication. This action also creates a considerable network load when the call throughput is high. If multiple proxies are used, the load could overwhelm the database or its network link. As a result, scalability is also affected.

The use of multiple databases (i.e., one local database per proxy) or adding more hardware resources to the database are not efficient solutions. As we described in Chapter 4, the effects of network latency could be reduced by a combination of parallelization and batching techniques. However, the network load to the database is still high enough to affect the scalability of the system. A more efficient approach is to reduce the number of queries to the database. To achieve this, we can use temporary authentication credentials that each proxy stores in memory and that can be used in multiple authentication operations without contacting the database. This approach reduces the load received by the database and the effects of network latency. Our proposed protocol follows this approach.

5.2 Proxychain Protocol Specification

In this section we present the details of the Proxychain protocol. We begin with describing the threat model. Next, we present protocol's design goals. We continue with a brief overview of hash chains, the basic building block of our construction. Finally, we provide a formal protocol definition.

5.2.1 Threat Model

Our scenario assumes a polynomial-time (PPT) adversary that has access to all the communications between a SIP proxy and its users. The adversary's goal is to send unauthorized requests to the proxy by masquerading as a registered user (i.e., user impersonation). For this purpose, the adversary can capture valid requests and resend them to the proxy (i.e., replay attack) or use them to forge new, arbitrary requests (i.e., session hijacking). In addition, the adversary can try to obtain users' authentication credentials from captured requests (i.e., offline dictionary attacks). Therefore, we assume that a strong password policy is actively enforced by the SIP provider.

We also assume that the database has a high level of security. Only trusted entities (i.e., proxies) are allowed to communicate with the database using a robust security protocol (e.g., TLS or IPsec). Therefore, threats against the database are not considered in our scenario. In addition, we do not consider attacks against the SIP UAs. In contrast, the proxies and the network traffic between proxies and UAs have a higher risk of being targeted by both active and passive attackers. Thus, we consider attacks against the proxies where an adversary can steal users' authentication credentials stored in the proxy.

Based on this threat model, the proposed authentication protocol should provide the following security guarantees. First, the proposed protocol should provide authenticity and integrity protection to each request send to the proxy. For this purpose, the

proposed protocol should rely on hash chains (see Section 5.2.3) and a Message Authenticated Code (MAC) primitive that meets the standard notion of unforgeability under chosen-message attack (UF-CMA [77]). Second, during a challenge-response authentication exchange, the proposed protocol should provide authenticity and integrity protection to both the proxy's challenge and the UA's response (i.e., mutual authentication). Third, the proposed protocol should be resistant to active threats such as replay and session hijacking attacks. Fourth, the proposed protocol should not leak information about the user's password or other private information. In particular, it should be resistant to offline dictionary attacks. Finally, the information stored in the proxy to authenticate the UA's requests should not allow an adversary to impersonate the user.

5.2.2 Design Goals

Proxychain design addresses some of the shortcomings of Digest authentication in SIP topologies with a centralized authentication service. Our first goal is *efficiency:* Proxychain should execute authentication operations faster than Digest authentication, allowing improved call throughput. Second, we focus on *scalability:* Proxychain should support more users and proxies than Digest authentication without the need for additional resources. In particular, Proxychain should reduce the bandwidth and processing time required by the database to avoid bottlenecks. Finally, our third goal is *security:* Proxychain should improve upon the security assurances provided by Digest authentication.

5.2.3 Hash Chains

A hash chain is created by applying a cryptographic hash function $H()$ (e.g., MD5, SHA-1, SHA-256) multiple times to a random value r to generate a sequence of values that can be used as one-time authentication tokens. A hash chain of length n is computed as:

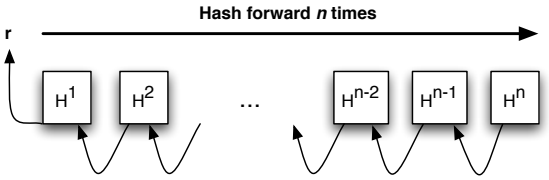


Figure 24: Hash chains are generated by hashing a secret value forward n times. A principal Bob stores the current value of the hash chain (H^c). A participant Alice can prove knowledge of the initial secret by presenting Bob with the previous value (H^{c-1}). If Bob hashes Alice's input and generates H^c, Alice must know the initial secret due to the one-way property of hash algorithms. Bob then makes the current value H^{c-1} and waits for Alice to provide H^{c-2}.

$$H^n(r) = H(\ldots H(H(r))\ldots)$$

Figure 24 provides a visual representation of this mechanism. Hash chains rely on the preimage resistant (i.e., one-way) property of cryptographic hash functions. When attempting to authenticate to a server possessing $H^n(r)$, the client transmits $H^{n-1}(r)$. The server then hashes $H^{n-1}(r)$ a single time and, if the result matches $H^n(r)$, authenticates C based on the computational infeasibility of an adversary guessing the correct preimage.

Note that there are a number of potential weaknesses with this basic construction. Specifically, hash chains do not provide mutual authentication as the server only verifies the client. Accordingly, an attacker can potentially impersonate the server and fool the client into computing a response for a low value of n (e.g., $n = 1$), allowing the attacker to recover all the remaining unused values (small "n" attack [106]). We address these concerns in our construction.

5.2.4 Design and Formal Description

Proxychain is designed to reduce the impact of latency and load on the remote authentication service by caching temporary authentication credentials at the proxies. Using hash chain-based credentials of length l, a proxy can authenticate multiple requests from a particular user with only $\frac{1}{l}$ queries to the database. The database

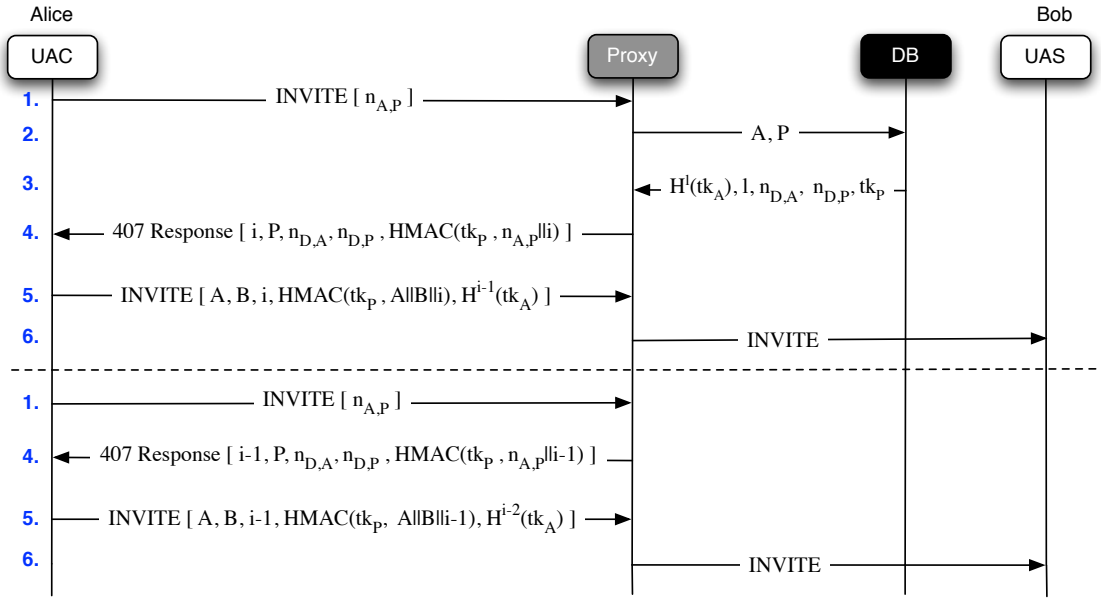


Figure 25: Call setup flow using Proxychain. For the first request (above dashed line), the proxy must request a temporary credential from the database. Subsequent requests (below dotted line) can be dealt with immediately by the proxy.

creates credentials based on the secret it shares with each user and determines the credential's parameters, including length, hash function, and expiration time. This approach is more secure than the associated Digest authentication mechanism, as the shared secret between the database and the user is never exposed to the proxies. A compromise of one of these servers, therefore, does not necessarily require password resets for large number of users.

Each proxy provides services only to users that are geographically close to it (i.e., based on IP address or ZIP code information), much like a traditional telephony switch. Each proxy accordingly needs to store credentials for only a subset of the total number of users in the system. We explore the overhead associated with such credential storage in Section 5.3.2.

Figures 25 and 26 provide graphical and formal definitions of the Proxychain protocol, respectively. A user Alice attempts to call Bob by first sending an INVITE request to her proxy, which contains the source and destination of the call and a

1. $A \rightarrow P : A, B, n_{A,P}$
2. $P \rightarrow D : A, P$
3. $D \rightarrow P : H^l(tk_A), l, n_{D,A}, n_{D,P}, tk_P$
4. $P \rightarrow A : i, P, n_{D,A}, n_{D,P}, \text{HMAC}(tk_P, n_{A,P}||i)$
5. $A \rightarrow P : A, B, i, \text{HMAC}(tk_P, A||B||i), H^{i-1}(tk_A)$

A, B, P, D	:	Alice, Bob, Proxy, Database
pwd	:	Shared secret (i.e., password) between Alice and database
$n_{D,A}, n_{D,P}, n_{A,P}$	:	Nonces
l	:	Hash chain length
i	:	Hash chain current sequence number
$H^i(x)$	:	*i-th* hash value of x, H(H(...H(x)...))
HMAC(k, x)	:	HMAC with key k on x [18, 112]
tk_A	:	PBKDF1$(pwd, n_{D,A}||P, c, len)$
tk_P	:	PBKDF1$(pwd, n_{D,P}||P, c, len)$
PBKDF1	:	Password-Based Key Derivation Function with c iterations and *len* bytes of output [105]

Figure 26: *Proxychain protocol:* The formal definition of the Proxychain protocol. We assume that there exists an encrypted channel (e.g., IPsec connection) between the proxy and the database.

nonce $n_{A,P}$ (Message 1). The proxy checks to see if it has a credential for Alice and, if not, queries the authentication database with the identifiers corresponding to Alice and the proxy (A, P) for a new hash chain (Message 2). Note that requests between proxies and the authentication database occur over a long-lived, encrypted and authenticated channel such as IPsec or TLS/SSL. The database generates a five-tuple that includes a new hash chain $(H^l(tk_A))$, the length of the hash chain l, nonces for both the proxy and Alice ($n_{D,P}$ and $n_{D,A}$), and a session key tk_P (Message 3). The hash chain secret tk_A and the session key tk_P are derived from the password *pwd* Alice shares with the database. To offer resistance to offline dictionary attacks, a Password-Based Key Derivation Function (PBKDF1 [105]) is used with salt values $n_{D,A}||P$ and $n_{D,P}||P$ correspondingly. The number of iterations of the PBKDF1 (c)

and the length of its output (*len*) are domain dependent parameters.

After receiving the tuple from the authentication database, the proxy returns a `407 Proxy Authentication Required` SIP message to Alice. This message includes a counter $i \leq l - 1$, the proxy's identifier P, the two nonces generated by the authentication database ($n_{D,P}$ and $n_{D,A}$) and a network authentication token based on an HMAC function [18, 112]. The authentication token is computed as HMAC$(tk_p, n_{A,P}||i)$ (Message 4). The client receives the response and uses Alice's password (*pwd*) to calculate the session key tk_P and then authenticates the message from the proxy. If the message authenticates properly, Alice then generates her session key tk_A and hashes it $i - 1$ times to generate $H^{i-1}(tk_A)$. Alice responds to the proxy by sending a new INVITE message containing A, B, i, $HMAC(tk_P, A||B||i)$ and $H^{i-1}(tk_A)$, which the proxy hashes forward a single time (assuming that the HMAC properly verifies) (Message 5). If $H(H^{i-1}(tk_A)) = H^i(tk_A)$, then the proxy records $H^{i-1}(tk_A)$ as the next legitimate credential, decrements i and the INVITE request is forwarded to Bob (message 6). On subsequent authentication attempts by Alice where $c < i - 1$, the proxy responds to Message 1 with Message 4, which contains $c, P, n_{D,A}, n_{D,P}$, HMAC$(tk_P, n_{A,P}||c)$.

Note that unlike Digest authentication, Proxychain provides mutual authentication. Specifically, the network authentication token HMAC$(tk_p, n_{A,P}||i)$ can only be produced with knowledge of tk_P and using the nonce supplied by the user Alice. Moreover, because only the user and the authentication database could have created tk_P (because only they have knowledge of shared secret *pwd*), an adversary can not create legitimate hash chains without the assistance of the authentication database.

5.3 Experimental Setup

In this section, we describe the experimental testbed and methodology we use to characterize Proxychain. We also discuss some of the implementation issues of our

protocol.

5.3.1 Testbed

Our experimental testbed is based on the VoIP infrastructure depicted in Figure 1, Section 2.2.2. As this figure shows, the testbed is composed of three main components: the authentication database, SIP proxies and the user clients (UAs). The database and proxies are run on servers from the Georgia Tech Emulab testbed.[1] We use seven servers to represent the infrastructure (one database and six proxies). These servers run Linux Kernel 2.6.26 (Fedora Release 8), have two 2.80 GHz Intel Xeon processors and 512 MB of memory. The UAs are run on servers from our research lab. A total of nine servers are used, each running multiple UA instances to generate call traffic. These servers run Linux Kernel 2.6.24 (Ubuntu 8.04.2), eight (8) 2.00 GHz Quad-Core AMD Opteron processors and 16 GB of memory.

The network latency between the proxies and the database is simulated using Emulab's traffic shaping functionality. In order to use realistic latency values, we performed measurements using the Planetlab network testbed.[2] Using the ping network tool, we measured the round-trip time (RTT) between a Planetlab node located in the University of Kansas and Planetlab nodes located at UC Berkeley (67.6 ms), Georgia Tech (33.1 ms), MIT (44.7 ms), Princeton (43.8 ms), the University of Texas (20.6 ms), and the University of Washington (43.4 ms). The RTT data was collected during a 24 hours period and average values were calculated. Finally, no additional latency values were simulated between the proxies and the UAs (latency was around 1 ms). The reason is that our testbed assumes physical proximity and low latency values (e.g., < 10 ms) between the UAs and the proxies. Simulating this latency is not necessary because it would not affect the test load generated by the UAs and our results (it would slightly affect the setup time of each call).

[1] `http://www.netlab.cc.gatech.edu/`
[2] `https://www.planet-lab.org/`

The proxies are implemented using OpenSIPS[3] 1.5.2. OpenSIPS is a mature open source SIP proxy optimized for high performance. The proxies are configured with minimal functionality (stateless configuration and basic modules required for routing). We run MySQL[4] 5.0.45 as our database, a well-known open source relational database management system. MySQL is run with a default configuration (no optimizations). Finally, SIPp[5] 3.1 is used to generate the UAs' workload, which conforms to a uniform random distribution. SIPp is an open source traffic generator for the SIP protocol. A total of 36 SIPp instances are used in our experiments (18 UACs and UASs). Default SIPp scenarios are modified to support INVITE and BYE authentication for Digest and Proxychain authentication (SIP call flows in Figure 3, Section 2.2.3 and Figure 25).

Each proxy serves requests for 200,000 unique users. The number of users per proxy is limited by the proxy's available memory, disk space in the database and the size of authentication credentials (see Section 5.3.2). As a result, the total number of users in the database is 1,200,000. All the users are part of a single SIP domain (no inter-domain calls). The experiments are executed and controlled remotely using a combination of Bash and Perl scripts. Metrics are gathered during the experiments using several well-known open source tools (e.g., top, mpstat, and vmstat). Call statistics are collected by the SIPp UAS instances.

5.3.2 Proxychain Implementation

Implementing Proxychain requires a combination of new code modules and modifications to existing software. In the proxies, OpenSIPS ($\approx$ 320000 lines of code (loc)) required approximately 710 loc to support Proxychain. In the UAs, SIPp ($\approx$ 3000 loc) required around 140 loc. In the database, we built a separate concurrent-process

[3] `http://www.opensips.org/`
[4] `http://www.mysql.com/`
[5] `http://sipp.sourceforge.net/`

server application to handle queries from proxies and the associated cryptographic operations. This server application required approximately 880 loc. The MySQL database software itself was unmodified. All of our experimental code, which was written in C, and supporting scripts are available at `http://www.cc.gatech.edu/~idacosta/proxychain.html`

Proxychain uses the same SIP headers in the challenge and response messages. For example, a Proxy-Authenticate header (challenge) looks as follows:

```
Proxy-Authenticate:PC realm="CISEC", i="10",
nda="0ec497d9a5ba5e1f2b2177d83fb3d341",
ndp="f1e992583dd5daecddea3309a01e5347",
hmac="15f5d33206e79eaea7245682d9953164"
```

where *PC* indicates the use of the Proxychain protocol, *realm* is proxy's identifier, *i* is the sequence number, *nda* and *ndp* are the nonces and *hmac* is the network authentication token.

The corresponding Proxy-Authorization header using Proxychain looks as follows:

```
Proxy-Authorization:  PC username="0000001",
realm="CISEC", i="10",
response="a0843d4b8a712284ff5a6fcd136c4b47,"
hmac="f9fd4ef6689850406a560965a4381c57"
```

where *response* is the next value in the hash chain sequence. The other parameters have the same meaning as in the Proxy-Authenticate header.

Our Proxychain implementation uses the MD5 hash function in order to compare it more directly and fairly to Digest authentication[6]. Nevertheless, our code requires

[6]Note that reported MD5 collision vulnerabilities [195] do not affect the security of hash chains and HMAC functions [190].

few modifications to support SHA-1 or SHA-256. With MD5, the size of a temporary authentication credential is 134 bytes. As a result, each proxy in our testbed requires a minimum of 26 MB of free memory for serving 200,000 users. In addition, for a direct comparison to Digest authentication, the PBKDF1 function was evaluated with only one iteration ($c = 1$) and an output length of 16 bytes. The use of a large number of iterations to defend against offline dictionary attacks will only add additional processing time in the database and the UAs.

5.3.3 Methodology

We perform a number of different experiments in order to characterize Proxychain. We specifically compare our protocol against a system with no authentication mechanism and one using Digest authentication. We do not measure more computationally expensive mechanisms such as TLS/SSL as previous studies have demonstrated that they provide significantly lower throughput [134, 130, 41, 34]. We collect the following metrics in most of our experiments: call throughput, message retransmissions, failed calls, bandwidth utilization and database CPU utilization. These are global metrics, the totals for the whole infrastructure (i.e., the call throughput is equal to the sum of the call throughput measured in each proxy).

The *call throughput* refers to the number of successful calls per second (cps) measured every five seconds. *Message retransmissions* corresponds to the number of SIP messages retransmitted due to the expiration of timers in the UAs. Our tests use the default retransmission time defined by SIP standards (500 ms). *Failed calls* refer to the total number of unsuccessful calls measured in the last period. In our experiments, we consider only call failures due to maximum number of retransmissions (maximum number of UDP retransmissions attempts has been reached). We use the default values in SIPp for the maximum number of retransmissions: five for INVITE messages and seven for others. Finally, *bandwidth utilization* corresponds to the total

Table 3: Response computation time at the UA and verification time at the proxy for Digest and Proxychain authentication. Proxychain adds little overhead to the response computation and it is more efficient performing verifications.

Protocol	Digest	Stdev	Proxychain	Stdev
Response (μsec)	116.81	13.59	184.76	49.92
Verification (μsec)	197.24	21.51	66.97	15.07

Table 4: Time required by the database to compute credentials with different hash chain lengths. For lengths < 100, the overhead is small.

Length	10	100	1000	10000
Time (μsec)	294.10	335.15	1383.53	11875.71
Stdev (μsec)	18.42	15.28	18.07	120.44

network throughput (KBytes/sec) measured from the database during each test.

During our experimental analysis, each test was run at least 10 times to ensure the soundness of the results. Average values are used in our analysis and a 95% confidence interval is provided in most of the graphs. Note that these bounds are often difficult to observe in our graphs as the values are generally very close to the mean.

5.4 Experimental Results

5.4.1 Microbenchmarks

To understand the computational differences between Digest and Proxychain authentication, we measure the time to compute a response in the UA and the time to verify a response in the proxy. To measure these values, we use network traces (100 samples per value). For Proxychain, the measurements are performed the first time a credential is used (hash chain length of 10). This corresponds to the worst case for response computation because it requires the highest number of hash operations (9 operations).

Table 3 shows the results. The UA running Proxychain requires approximately 70 μsec of additional computation than one running Digest authentication. This difference is due to the additional integrity checks and hash operations required by

Proxychain in the UA. However, this difference is not significant as individual UAs do not perform large amounts of computation in this system. Interestingly, the response verification is nearly three times faster when Proxychain is used by the proxy. The reason is that Proxychain only requires two hash operations to verify a response. On the contrary, Digest authentication requires three hash operations and additional checks to verify a response. Based on these results, we argue that the computational overhead added by Proxychain is not significantly different from the one added by Digest authentication.

We also evaluate the overhead of generating hash chains of varying lengths. Specifically, we measure the time required by the authentication database to generate credentials of lengths 10, 100, 1000 and 10000. As before, we use network traces to measure the time for each configuration (100 samples per configuration). Table 4 shows the results of these experiments. As expected, increasing the hash chain size increases the time required to generate credentials. The additional time remains small for hash chains with length up to 100 (< 350 μsec).

5.4.2 Call Throughput

Microbenchmarks provide insight into the overhead that can be expected at each component of the network. However, they do not provide a picture of the overall behavior of a system. Accordingly, we characterize the interaction of those components by measuring total call throughput. We compare throughput for systems configured to use Digest authentication, Proxychain and no authentication mechanism. UAs generate an increasing call load (270 cps increments every 5 seconds) over the course of 10 minutes. In addition, we evaluate the best configuration for each protocol. For Digest authentication, we use close to 100 concurrent proxy-processes per proxy. For Proxychain, we preload each proxy with all its user credentials (200K credentials with hash chain length of 10) before each experiment and use 8 concurrent proxy-processes

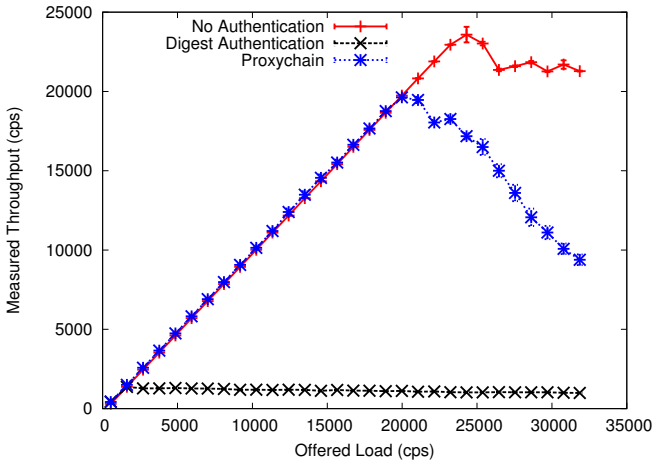


Figure 27: Total call throughput for no, Digest and Proxychain authentication. Proxychain's maximum call throughput is close to the one obtained without authentication.

per proxy (OpenSIPS recommended value).

Figure 27 shows the results of these experiments. Without authentication (baseline configuration), the network supports a maximum call throughput of nearly 24,000 cps. When Digest authentication is used, the maximum call throughput drops dramatically to approximately 1,160 cps. This result represents a 95% reduction in call throughput when compared with the baseline configuration. For Proxychain, the result is more favorable: a total call throughput of over 19,700 cps. In this case, the call throughput drops by only 18% when compared with the baseline configuration. However, when compared to Digest authentication, Proxychain allows an increase of over 1,700% (more than an order of magnitude). Accordingly, Proxychain is significantly more efficient than Digest authentication in this architecture.

Figure 28 provides insight into the poor performance of Digest authentication. The database process rapidly reaches 175% CPU utilization (dual-core machine). This behavior indicates that queries from the proxies saturate the authentication database, making it a bottleneck. We observe the opposite when using Proxychain. The database was virtually idle ($< 5\%$ CPU utilization) before the system reaches its maximum call throughput, at which point the system becomes unstable due to the

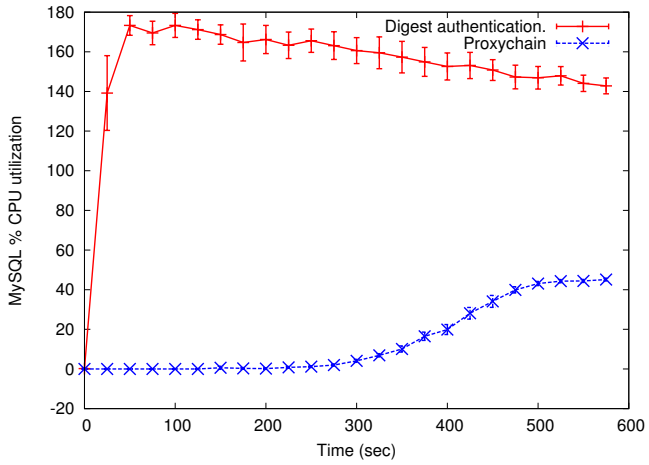


Figure 28: Percentage of CPU required by the database process for Digest and Proxychain authentication. The database process is virtually idle when Proxychain is used.

high number of retransmissions.

A naïve solution to improve Digest authentication performance would be to use a more powerful database. Therefore, we repeated the experiment using a quad-processor server for the database. As expected, the maximum call throughput increases, but only to approximately 4,000 cps. However, in this experiment the database does not saturate – CPU utilization is below 300%. In this case, throughput fails to increase further due to the network latency between the proxies and the database.

Another important difference is the total bandwidth required for both configurations. The message overhead between a UA and the proxy are arguably equivalent. Message 4, the challenge, requires an additional 92 B and 165 B for Digest and Proxychain authentication, respectively. The response in Message 5 similarly requires an additional 199 B and 153 B. At its maximum call throughput (measured from the database), Digest authentication required almost 130 and 430 KBytes/sec for queries and responses respectively. In contrast, Proxychain required less than 1 KByte/sec for both, queries and responses. As expected, the use of temporary credentials significantly reduces the total number of queries to the database.

The previous results also mean that increasing the hash chain length (>10) will not help to improve performance in our testbed. The reason is that the load in the database is already low with a hash chain length of 10. Using a longer size will make the load even lower but the difference will not affect the overall performance of the system. On the contrary, using hash chains that are too long could affect performance because of the additional hash operations that will be needed by the UACs and the database.

Finally, for the baseline and Proxychain configurations, the maximum call throughput is limited by the proxy application itself: OpenSIPS. Analyzing the resources usage statistics (memory, CPU and bandwidth) collected during the experiments for the different testbed components, we find that none of the resources are completely used (no shortage of resources) when the two configurations reach the maximum call throughput. Based on this evidence and in our experience with OpenSIPS, we can conclude that the OpenSIPS software is the performance bottleneck for no authentication and Proxychain configurations. Using an optimized version of OpenSIPS or a faster proxy server application will provide higher call throughput values.

5.4.3 Scalability

In this set of experiments, we evaluate how the testbed handles an increasing number of users, and therefore, an increasing load. To simulate a varying number of users, we measure performance with a varying number of proxies, where each proxy represents 200,000 users. Using a similar procedure as in the previous test, we measure the call throughput for 3, 4, 5 and 6 proxy configurations (600K, 800K, 1M and 1.2M users respectively).

The results are presented in Figure 29. We can see that for Digest authentication, the maximum call throughput measured is approximately the same (≈1,200 cps; linear

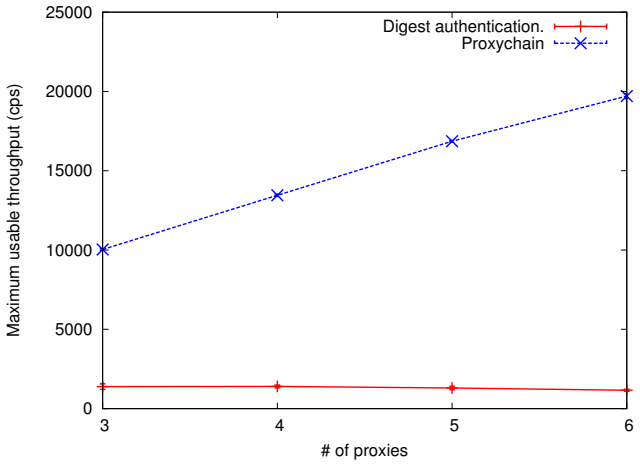


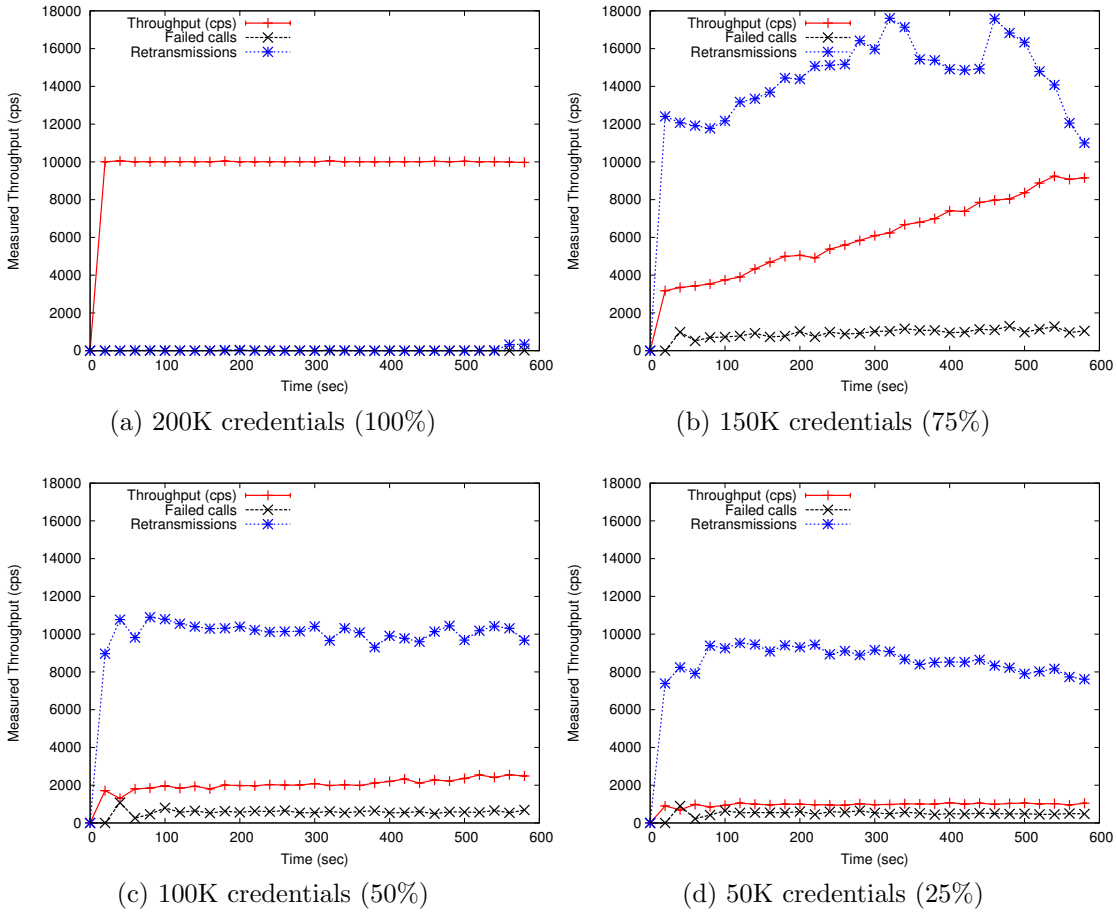
Figure 29: Throughput measured for a range of proxies using Digest and Proxychain authentication. Proxychain is considerably more scalable than Digest authentication.

regression: $y = -79.6x + 1670.5$, $R^2 = 0.848$ [7]) for all the configurations. The reason is that even for a three-proxy configuration, the database becomes saturated rapidly (see previous test). Therefore, Digest authentication limits the scalability of the system. For Proxychain, the maximum call throughput increases linearly with the number of proxies (≈3,250 cps per proxy; linear regression: $y = 3243.9x + 416.5$, $R^2 = 0.998$). From these results, we can conclude that Proxychain allows the system to grow by just adding new proxies and without requiring changes to the database.

5.4.4 Credential Preloading in the Proxies

In the previous tests, we evaluated Proxychain's performance using a best-case scenario: each proxy had all the credentials in memory before the tests started. We now evaluate performance when a lower number of credentials are preloaded in each proxy. For this purpose, we use a similar procedure as in previous tests but with two exceptions. First, we use a constant workload of 10,000 cps with no ramp-up period. Second, we preload the proxies with 200K, 150K, 100K and 50K credentials in each test.

[7] R^2 is the correlation coefficient, which indicates goodness of fit, with 0 being no match and 1 being perfect.



(a) 200K credentials (100%)

(b) 150K credentials (75%)

(c) 100K credentials (50%)

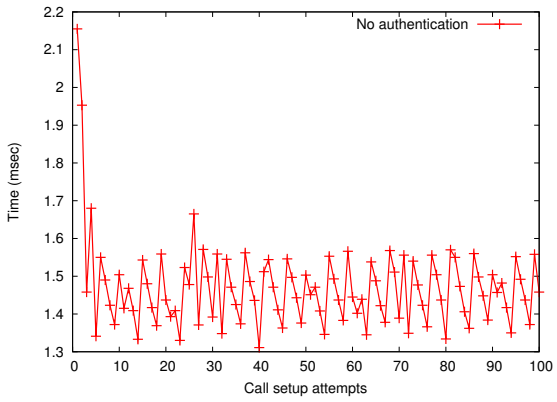
(d) 50K credentials (25%)

Figure 30: Call throughput measured for different number of credentials preloaded in the proxies and a constant offered load (10K cps). Proxychain requires that proxies have most of the credentials in memory for maximum performance.

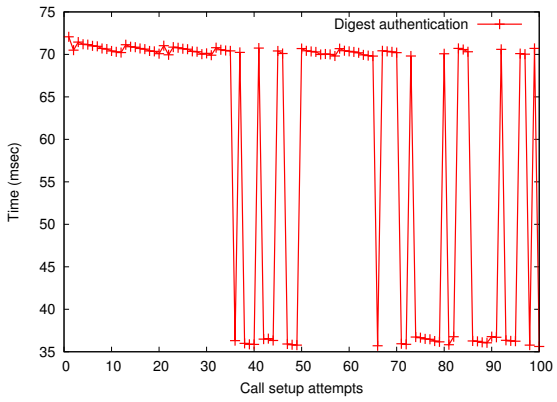
Figure 30 shows the results for all the configurations. For the 200K configuration (best-case, Figure 30a), the call throughput reaches 10,000 cps quickly (< 10 sec) with virtually no message retransmissions or failed calls. For the 150K configuration (Figure 30b), the call throughput jumps to approximately 3,000 cps, and then continues increasing until it reaches almost 10,000 cps by the end of the test. However, a large number of retransmissions and failed calls occur. Finally, for the other two configurations (Figures 30c and 30d), the behavior is worse. The maximum call throughput measured was around 2,000 and 1,000 cps respectively during the experiments. The number of retransmissions and failed calls is also constantly high. In theory, each

Figure 31: Call setup time for four different configurations: no, Digest, Proxychain and Proxychain with prefetching authentication. The call setup time for Proxychain with prefetching is similar to the one obtained with no authentication.

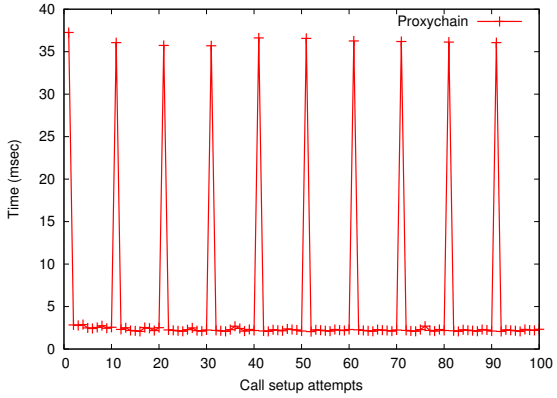
(a) No authentication



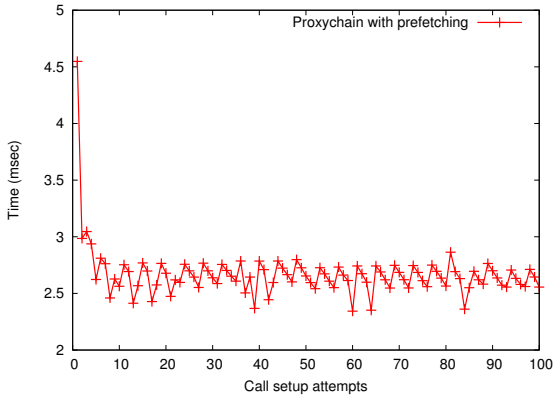
(b) Digest authentication



(c) Proxychain



(d) Proxychain with prefetching



configuration should have reached 10,000 cps after some period of time. However, the large number of retransmissions makes the system unstable. These results show the importance of having most of the credentials stored in the proxies to avoid the negative effects of retransmissions, especially when high loads are expected.

5.4.5 Prefetching Mechanism

The previous test shows that Proxychain is more effective if each proxy has credentials for almost all its users (best case scenario). However, credentials are stored or updated in the proxy only after a user request that requires authentication. Therefore, we implement a prefetching mechanism that automatically queries the database for

credentials without requiring any user action. This mechanism, running as a separate proxy process, checks if a user has a credential in the proxy or if her credential has already expired (i.e., $l = 0$). In short, the prefetching mechanism guarantees the best case scenario for Proxychain.

In this experiment, we characterize the effect of the prefetching mechanism on the call setup time for individual UAs (time elapsed between the first INVITE request and the 200 OK response). We use a UA sending a low load (< 5 cps) to a single proxy and estimate the call setup time using network traces (100 samples). Four proxy configurations are used: no authentication, Digest authentication, Proxychain and Proxychain with prefetching.

Figure 31 shows the results for each configuration. As expected, when no authentication is used (Figure 31a), the call setup is the fastest: 1.47 ms on average. For Digest authentication (Figure 31b), we can observe the effects of the RTT between the proxy and the database ($\approx$33 ms) on the call setup time. Two call setup times are measured: 36 and 71 ms approximately. The reason is that for the first value, only one RTT is required during call setup, while for the second value, two RTTs are required due to the low test load used (no TCP piggybacking). In general, only one RTT is required, so we can assume that the call setup time for Digest authentication is approximately 36 ms. In the case of Proxychain, Figure 31c shows how the temporary credentials reduced the call setup time while they are valid. While the credentials are active (hash chain size > 0), the call setup time is only 2.27 ms on average. Once a credential expires (hash chain size $= 0$), a query to the database is required, so the call setup time increased by one RTT: 36.28 ms on average. When Proxychain is used with prefetching (Figure 31d), the average call setup time is only 2.67 ms. The reason is that no credential updates are performed during call setups. Instead, credentials are updated by the prefetching process automatically, before they are required in a call setup. Therefore, the call setup time when Proxychain is used

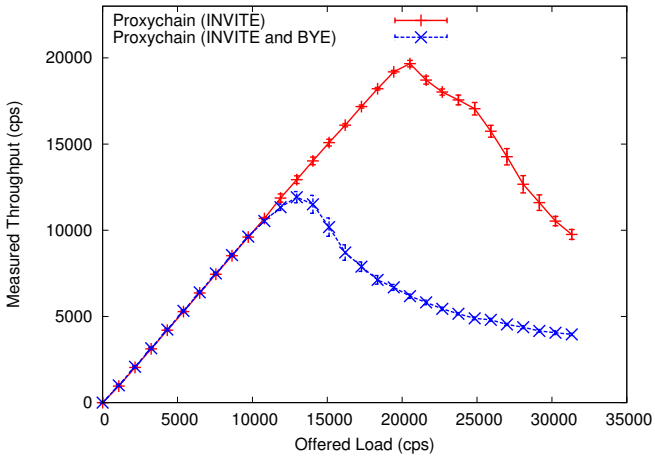


Figure 32: Throughput for INVITE and INVITE and BYE Proxychain authentication. Proxychain allows authentication of two requests per call while still supporting high throughput.

with prefetching is close to the call setup time when no authentication is used (≈1 ms difference). Accordingly, prefetching helps to eliminate the effect of network latency on call setup time.

5.4.6 Authenticating Multiple Message Types

In our final set of experiments, we explore the effect of authenticating multiple SIP message types request per session (call dialog). For example, the lack of authentication of BYE requests allows several reported attacks against SIP deployments [207]. However, if BYE requests are also authenticated using Digest authentication, the performance of the system will decrease even more due to the additional operations and queries to the database. In this experiment, we evaluate the impact of authenticating INVITE and BYE requests on performance when Proxychain is used. We use a similar procedure as in Section 5.4.2 (i.e., no prefetching). The only difference is that the proxies and UACs are configured to authenticate BYE in addition to INVITE requests.

Figure 32 shows the call throughput for the two configurations: INVITE and "INVITE and BYE" Proxychain authentication. As expected, the maximum call

throughput supported by the testbed decreases when two requests are authenticated to approximately 12,000 cps. This represents a performance drop of nearly 50%. The reason is that credentials are used faster (twice as fast) because two authentication operations are required per call, making the number of queries to the database increase, resulting in higher CPU and bandwidth utilization. However, the use of Proxychain to authenticate two types of signaling messages still provides over 800% greater throughput than Digest authentication authenticating a single message.

Finally, we test if increasing the hash chain length improves the performance in this scenario. The idea is that, if credentials are used faster when two requests per call are authenticated, increasing the hash chain length should reduce how fast they need to be replaced. This will result in lower load to the database and increased throughput. The experiment confirms our hypothesis: using a hash chain length of 20 results in a maximum call throughput of almost 14,000 cps. This represents an improvement of almost 17% when compared to using hash chain length of 10. However, increasing the hash chain length further does not improve performance. On the contrary, the performance drops back to almost 12,000 cps with a hash chain size of 30 (using a longer hash chain caused earlier retransmissions, which affects the performance). The reason for these results is that we again reach the limits of the proxy application. The call throughput achieved is lower because the authentication of two requests involves additional messages and operations.

5.5 Discussion

5.5.1 Performance

The results presented in the previous section show that Proxychain effectively addresses the limitations of Digest authentication in VoIP topologies with a centralized authentication service. Specifically, Proxychain reduces the effects of network latency, allowing higher throughput. In our testbed, Proxychain's performance improvement

was enough to reach the limits imposed by the proxy application (OpenSIPS). Moreover, Proxychain reduces the load received by the database, improving scalability.

The caching of temporary authentication credentials across the proxies allows our solution to perform so much better than Digest authentication. Not surprisingly, cellular networks perform a similar distributed caching of credentials, which are generated by a *Home Location Register* (HLR) and stored in the Mobile Switching Center/Visitor Location Register (MSC/VLR) closest to the client. However, the Proxychain approach is more efficient in terms of memory. Specifically, the current approach used in cellular networks requires that multiple credentials are stored in the MSC/VLR per user. Should the authentication database (HLR) wish to reduce its load, the proxies (MSC/VLRs) would need to be equipped with additional memory. Because Proxychain authentication credentials require a *constant* amount of memory regardless of the hash chain length, our approach is also more scalable than traditional caching. This property is particularly advantageous as it allows for more dynamic behavior by the infrastructure. For example, a database could monitor the received load and automatically increase the length of the hash chains in response to a spike in the load (e.g., busy hours, DoS attack or a flash crowd). We plan to explore such dynamic reprovisioning in future work.

The performance gains obtained in our experiments are based on the assumption that each proxy has most of its users' credentials most of the time. We also assumed that each proxy has a fixed set of registered users and that users do not register with other proxies often (e.g., traveling to another state). These assumptions can be relaxed by providing additional cache space in the proxies. For example, each proxy will have a cache of fixed size, and keep in the cache the credentials of the most active users. When new users register with a proxy, the proxy can use an eviction policy to replace the credentials in the cache based on frequency of use. In this way, each proxy could handle a variable number of users (more flexibility). This approach will

be evaluated in future work.

The call throughput numbers achieved in our testbed could be considered high for commercial VoIP deployments. For example, AT&T average nationwide call volume is estimated to be around 300M calls per day, or an average of 3,472 cps [70], or roughly 17% of the throughput provided by our architecture. We note that while our testbed lacks some of the other functionality that a provider may chose to deploy (e.g., billing, media gateways), the performance benefits provided by Proxychain represent a significant potential improvement to real networks. Specifically, the additional capacity offered by Proxychain can serve as a defense mechanism to handle unexpected increments of requests for service.

The performance gains obtained by Proxychain requires some trade-offs. First, a proxy using Proxychain requires to keep a small amount of state for all its users (credentials), which is not necessary for Digest authentication. However, our experiments demonstrated that this was not a significant burden. UACs also need to perform more authentication operations when Proxychain is used. Specifically, Proxychain requires additional integrity checks and hash chain computations are required to create a response. Nevertheless, the most expensive operations are hash computations that are in general very efficient to execute. In addition, the use of adequate hash chain lengths (i.e., < 100) and caching intermediate results in the UAC can reduce these overheads. Third, the database also requires to perform computation to create the user credentials. However, this is a one-time cost and it is lower than processing an equivalent number of requests per user as in Digest authentication.

In general, any SIP infrastructure with multiple proxies and a remote central authentication service will benefit from Proxychain, even if the performance requirements are not carrier-level. For example, the SIP infrastructure of a multinational corporation where each regional office has a SIP proxy and the central database is located in the headquarters. The use of Proxychain in this scenario will reduce the load

to the database (lower bandwidth and CPU utilization) and provide more security. As our results shows, the main requirement is to cache the credentials of most of the users (e.g., $> 75\%$) served by each proxy. This is not a hard requirement given the size of the credentials and the memory costs. Even in environments with high mobility requirements, caching the credentials of all the users in all the proxies or using caching algorithms are reasonable options. Finally, the concepts behind Proxychain can also be used in other domains with similar topology requirements. For example, remote authentication services such as RADIUS or DIAMETER, or authentication in IP Multimedia Subsystem (IMS) deployments could benefit from the performance, scalability and security advantages offered by Proxychain.

5.5.2 Security and Threat Analysis

Proxychain provides the security guarantees described in Section 5.2.1. To provide authenticity and integrity protection, Proxychain uses one-time authentication tokens based on a hash chain and a HMAC function. The HMAC value in the challenge message (message 4 in Figure 25) protects the integrity of the hash chain counter (i). Moreover, this HMAC also provides proxy authentication because it demonstrates that the proxy knows the session key (tk_P). In the response message (message 5 in Figure 25), another HMAC is used not only to protect the integrity of the hash chain counter but also the source and destination addresses of the SIP request. Therefore, even if an adversary intercepts a response message and prevents it from reaching the proxy, the adversary cannot modify it to authenticate a different SIP request.

User authentication is not only achieved with the HMAC but also by including the next unused hash chain value ($H^{i-1}(tk_A)$) in the response message. These values authenticate the user because they demonstrate knowledge of the user's password. Due to the one-way property of cryptographically secure hash functions, it is computationally infeasible for an adversary to reverse the hash function and obtain an earlier

value of the chain or the chain's secret (tk_A). Moreover, HMAC is UF-CMA secure, as any PRF (Pseudo-Random Function) MAC is also UF-CMA [17] (HMAC was proven to be a PRF [16]). UF-CMA security provides strong guarantees that an adversary with reasonable resources will not be able to forge a valid HMAC without knowledge of the user's password or the session key. In is important to note that, while hash functions such as MD5 and SHA-1 are no longer considered secure due to reported collision attacks [195, 194], they can still be used in HMAC functions because HMAC does not require collision resistance for its formal security proof [16, 190]. However, it is recommended to use a more robust underlying hash function such as SHA-256 in new protocols, as MD5 and SHA-1 support will decrease.

As described before, Proxychain provides mutual authentication. Proxychain authenticates not only the proxy's challenge (server authentication) but also the UA's response (user authentication). This property provides stronger resistance to threats such as replay, server spoofing and man-in-the-middle attacks. In addition, the mutual authentication guarantees provided by Proxychain are less expensive and easier to implement than the ones provided by protocols such as TLS or IPsec.

To prevent session hijacking attacks, each authentication token (i.e., HMAC and hash chain value) are tied to a particular user request; thus, an adversary cannot modify or use a valid token to authenticate arbitrary requests. Proxychain also prevents replay attacks of both the challenge and the response message. The freshness of the proxy's challenge is guaranteed by the nonce $n_{A,P}$, included in the initial user request (message 1 in Figure 25). The freshness of the UA's response is guaranteed by the hash chain value attached. Once a hash chain value is successfully validated by the proxy, it cannot be used again; thus, preventing replay attacks.

To mitigate the risks of offline dictionary attacks, Proxychain derives the session key (tk_P) and hash chain secret (tk_A) from the user's password via a Password-Based Key Derivation Function (PBKDF1 [105]). Such a function performs a large number

of cryptographic operations (e.g., hash or HMAC) to derive a key from the user's password. The additional computation time reduces the speed of offline dictionary attacks; thus, making such attacks more difficult. The number of operations required by the PBKDF varies according to the scenario and adversarial model. Moreover, the use of salt values in the PBKDF prevents the use of precomputed tables of hash values (i.e., rainbow tables) commonly used in dictionary attacks. In addition, the SIP provider should enforce strong password policies to provide additional defense against dictionary attacks.

An active attacker could try to compromise a proxy and steal its cached credentials. However, *Proxychain credentials cannot be used to impersonate users* (another advantage of hash chains). Instead, stolen Proxychain credentials could only be used to impersonate the proxy to the users due to the session key included in the credentials. In this scenario, only mutual authentication will be affected, resulting in the same security level provided by Digest authentication or S/Key (no server authentication). Therefore, an attacker will still need considerable effort to impersonate users even if she manages to steal the credentials cached by the proxy. While not implemented in our testbed, Proxychain can also include a revocation mechanism where the database can invalidate the credentials cached in a proxy. This mechanism will be useful in situations where a user needs to change her password or when a proxy has been compromised.

Proxychain not only offers the security advantages of hash chains protocols (i.e., protection against eavesdropping and replay attacks), but also solves some of the weaknesses associated with these protocols [135]. For example, as mentioned before, Proxychain protects the integrity of the hash chain counter in both the challenge and response messages. This feature protects against an attacker located between the proxy and the client trying to change the counter (i) in the challenge to a lower value ($i = 1$) to obtain the complete hash chain sequence (i.e., small n attack [106])

In addition, Proxychain does not require hash chain synchronization[8] as S/Key does. The reason is that the hash chains are generated based on a secret derived from users' passwords.

Finally, Proxychain makes SIP authentication cheap enough to authenticate more than one message per session. Authenticating more SIP messages per session provides protection against several known attacks that target current SIP deployments. From the security perspective, all the messages should be authenticated to avoid vulnerabilities. Proxychain represents a first step in this direction.

5.5.3 Availability

The availability of the database is critical in scenarios with a central authentication service. For example, if the database becomes unavailable, the proxies will be unable to authenticate UAs requests. As a result, no call sessions can be established until the database is back online. This risk can be mitigated through mechanisms such as high availability clusters or backup sites. However, these alternatives are typically expensive and complex to manage.

Proxychain offers a cheaper alternative for database outages. The idea is that the database can create a list of authentication credentials with long enough hash chains and no expiration time. These backup-credentials can be stored offline in each proxy location and be activated when the database is not available. Once each proxy loads the backup-credentials in memory, they will be able to authenticate UA requests as long as the credentials are active (sequence counter > 0). A naïve approach would be to generate backup-credentials with uniformly long hash chains (i.e., length = 1,000) to reduce the risks of users finishing their credentials before the database is back online. However, this approach is inefficient because very long hash chains will cause unnecessary overheads in the database and the UAs and lower performance

[8]Setting a new hash chain once the current one expires, using a secure secondary channel.

during their generation. A more efficient approach would be to estimate the necessary length of the hash chains based on the expected time that the database is going to be unavailable. For example, a provider needs to install new hardware, requiring the database to be offline. The provider can estimate how many authenticated requests occur in a period of six hours based on its call statistics. For example, the provider can determine the call rate of its most active users. Assuming that the most active users make 10 calls per hour during busy hours, backup-credentials with a hash chain length of at least 60 will be required (also assuming that only one request per call is authenticated). Using Table 3, we know that the time to compute one credential with hash chain length = 100 is approximately 335 μsecs. Therefore, if the provider has 5 million users, the database will require approximately 28 minutes of computation to generate backup-credentials that will be active during 6 hours. This simple calculation could be made more robust by identifying those users most likely to far exceed the uses of the temporary credentials (i.e., profiling via long-term logging) and selectively increase the length of their hash chains.

5.6 Summary

VoIP has and will continue to change telephony. These systems not only drastically reduce the costs associated with building and providing such services, but also offer the potential for rich new sets of features. Unfortunately, the large-scale usage of VoIP also creates a number of new security concerns. In this chapter, we develop Proxychain, a mechanism that provides strong authentication between VoIP providers and their customers. Unlike previously deployed mechanisms, Proxychain is highly scalable and offers throughput improvements of greater than an order of magnitude. This increased efficiency allows providers not only to support a much larger customer base on a relatively limited hardware footprint, but also increases the overall security of the network by allowing for multiple message types to be authenticated. In so

doing, we have significantly increased the robustness of VoIP systems.

CHAPTER VI

ONE-TIME COOKIES: ROBUST AND EFFICIENT HTTP SESSION AUTHENTICATION VIA STATELESS AUTHENTICATION TOKENS

In Chapters 4 and 5 we studied the challenges associated with the development of robust authentication protocols for large-scale VoIP applications. In this and the following chapters we will explore similar challenges in large-scale Web applications (see Section 2.3), a more widespread class of Internet applications.

Web applications rely primarily on the HTTP protocol (see Section 2.3.2) to communicate with clients (i.e., browsers). Browsers use HTTP to requests and receive resources from the web application's servers (i.e., web servers). However, HTTP is a stateless protocol. Requests to a web server are treated as independent transactions with no relation to each other. While simple and scalable, this design is not adequate for web applications that require sessions – the association of multiple transactions to a single user (e.g., online banking and e-commerce applications). HTTP cookies (see Section 2.3.5), small pieces of data that keep session state information in the browser, were designed to address this limitation and rapidly became the dominant mechanism for HTTP session management.

Although cookies are a practical and efficient mechanism for session management, they introduce a number of security risks, especially when employed as session authentication tokens – a function for which they were not specifically designed [73]. For example, most web applications rely on the security provided by HTTPS to protect the user's password during the login process. During this step, the web application

generates cookies that the user can later employ as lightweight session authentication tokens. However, due to performance concerns, many web applications switch to HTTP after the user logs in and cookies are transmitted "in the clear". As a result, cookies are exposed to any adversary eavesdropping on the communication. Because cookies are static, an adversary can use them to gain unauthorized access to the user's session. While these session hijacking or "sidejacking" attacks are not new, a significant number of web applications are still vulnerable [187, 191]. Several factors such as the proliferation of open wireless networks and the release of automated attack tools [86, 31, 153, 110] have increased the risk of this threat. The most recommended defense is to use HTTPS to protect all communications with the web application ("always-on HTTPS"). However, deploying always-on HTTPS can be challenging due to performance and financial concerns, particularly for distributed systems. More importantly, always-on HTTPS is not a complete solution; cookies can still be exposed due to configuration errors [120] or by attacks against HTTPS [46] and the browser [79]. In short, always-on HTTPS does not address the root cause of the problem: *cookies are weak session authenticators.*

More robust alternatives to authentication cookies have been proposed [150, 24, 7, 39] (see Section 3.2.1). However, they have not been adopted due to their additional requirements and complexity. Specifically, most of these alternatives require state in the web server. This is a problem for highly distributed web applications because this state needs to be synchronized among servers in different geographic locations (see Section 2.3.1). Thus, the effect of network latency will not only make synchronization operations more expensive, but will also cause valid requests to be denied due to "out-of-sync" state. Web 2.0 applications are particularly affected by this problem due to their higher request concurrency. In short, *proposed alternatives to authentication cookies fail to address the operational requirements of highly distributed Web 2.0 applications and, as result, have not been deployed.*

In this chapter, we present One-Time Cookies (OTC), a more secure alternative to authentication cookies that does not require state in the web application. Instead of using a single, static token to authenticate each request, OTC generates a unique token per request based on a session key. Each OTC token is tied to a particular request by using a Hash-based Message Authentication Code (HMAC); hence, an adversary cannot reuse OTC tokens to illicitly redirect a session. To avoid state in the web application, OTC borrows the concept of Kerberos service tickets [142]. Like in Kerberos, an OTC session ticket contains the information the web application needs to verify an OTC token (i.e., session key), encrypted with a master key only known by the web application. Thus, any web application's server can verify OTC tokens without keeping any volatile data, *one of the main barriers for deploying alternatives to cookies in highly distributed systems.* Unlike cookies, OTC credentials are also securely stored and isolated from other browser components. We evaluate our proposed mechanism and demonstrate an overhead similar to the insecure traditional cookie approach. In summary, *OTC preserves the performance and scalability benefits of cookies while providing stronger security guarantees.*

We strongly believe that OTC raises the bar against real threats, but are careful not to over claim the guarantees that OTC can provide. Specifically, while our approach efficiently eliminates session hijacking attacks by ensuring session integrity (i.e., the integrity of navigation requests), it does not provide confidentiality or full integrity protection for the information exchanged between the browser and the web application. If these additional security guarantees are required, OTC can be used together with always-on HTTPS; *OTC and HTTPS are complementary security mechanisms.* OTC's main goal is to replace cookies as session authenticators, with a performance-conscious solution that can be deployed across traditional and highly distributed web applications. In so doing, we make the following contributions:

- **Designing and implementing a more secure and stateless alternative**

to authentication cookies: We identify key properties required to achieve a robust and practical alternative to authentication cookies. Based on these properties, we develop a protocol that prevents adversaries from successfully replaying captured authentication tokens to gain unauthorized control of a web session. Most importantly, our protocol does not require expensive state synchronization across the web application unlike previously proposed mechanisms, making it appropriate for highly distributed Web 2.0 applications. We implemented a proof-of-concept plugin for the popular blogging platform WordPress and extensions for both Firefox and Firefox for mobile browsers, demonstrating the deployability of our solution. However, we ultimately envision such mechanisms being included in the browser itself.

- **Conducting an extensive analysis on multiple platforms:** We perform extensive performance tests based on our OTC implementation, including desktop and mobile clients. Our experiments show that OTC and cookies have similar performance in both the web application and the browser. For example, the overall latency added by OTC to a WordPress page request was less than 6 ms when compared to cookies. We also apply ProVerif [21, 22] to formally verify the security properties of the OTC protocol.

- **Making our OTC implementation available to the community:** The WordPress OTC plugin and the OTC extensions for Firefox and Firefox mobile are already available online at: `http://www.cc.gatech.edu/~idacosta/otc/`. Any WordPress-based web site can incorporate OTC in matter of minutes and point their users to either the desktop or mobile Firefox extensions.

The remainder of this chapter is organized as follows: Section 6.1 offers important background information about session hijacking attacks and presents our motivation; Section 6.2 explains the design and formal description of OTC; Section 6.3 offers our

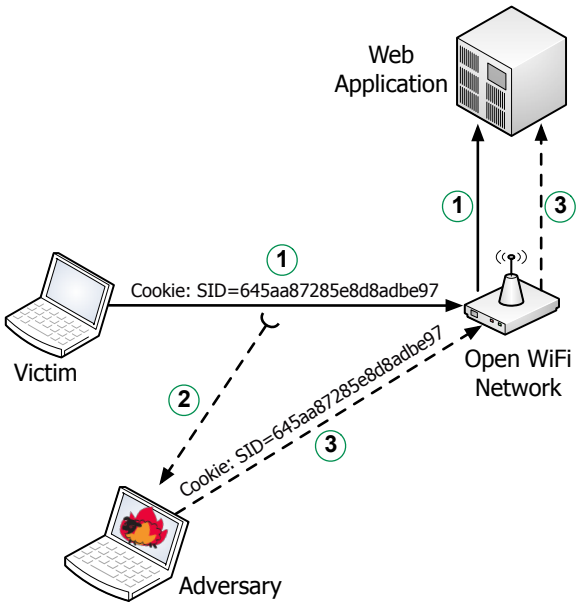


Figure 33: Simplified view of a session hijacking attack. (1) After login, the victim sends requests to the web application using a cookie for authentication. (2) Because this request is sent over HTTP, an adversary can eavesdrop the request and capture the cookie. (3) Finally, the adversary can use this cookie to send arbitrary requests to the web application, successfully hijacking the victim's session.

security analysis of OTC; Section 6.4 presents our experimental testbed, tests and results; Section 6.5 offers additional analysis and discussion of our proposed solution; Section 6.6 provides a summary of our work.

6.1 The Session Hijacking Threat

By design, HTTP cookies are static (see Chapter 2.3.5); they do not change during their lifetime. Hence, if an adversary steals authentication cookies, she will be able to impersonate the user associated with these cookies. This type of attack is known as *session hijacking* or *sidejacking* because the adversary takes control of the user's session.

Figure 33 shows a simplified view of a session hijacking attack. After logging in, the victim uses an authentication cookie (*SID*) on each request to the web application (step 1). As it commonly happens, the cookie is sent unprotected across the network and it is captured by an adversary eavesdropping on the communication (step 2).

For example, the adversary can use an automated tool such as FireSheep [31] for this attack. Finally, the adversary can use the stolen authentication cookie to make arbitrary requests to the web application as the user (step 3), until the cookie expires. It is important to note that the stolen cookie will remain valid even if the victim logs out of the web application. Cookies are stateless; therefore, the web application cannot revoke them (the web application could change the key(s) used to create the cookies, but that will revoke the cookies for all users).

Session hijacking attacks are not new; however, several factors have increased the risk of this threat. First, the increasing popularity and importance of web applications makes them a valuable target. Google, for example, was forced to improve the security of Gmail due to several incidents against its users in China [171, 35]. Second, the proliferation of wireless networks, particularly open Wi-Fi access points (e.g., airports, libraries, stores) increases the risk of these types of attacks. Third, the release of several automated, easy-to-use tools to perform session hijacking [86, 31, 153, 110] has brought session hijacking to the masses. For example, FireSheep [31], the most popular of such tools, has been downloaded almost two million times since its release in December 2010.

Cookies include an expiration time to reduce the window of opportunity for a session hijacking attack. However, many web applications use long expiration times (e.g., from hours to weeks if the "remember me" option is used) to avoid affecting user experience. This approach reduces the effectiveness of the expiration time against session hijacking. Cookies could also employ other alternatives to guarantee their freshness such as nonces (e.g., a challenge-response protocol) or counters (e.g., one-time password mechanisms such as HOTP [138]). While these mechanisms are more robust against session hijacking than timestamps (i.e., expiration time), they require additional messages per request or additional state in the web application.

Several alternative mechanisms have been proposed to defend against session hijacking attacks (see Section 3.2.1). However, most of these alternatives are not adequate for highly-distributed web applications, in particular due to the high costs of state synchronization in these applications (see Section 2.3.1). Instead, always-on HTTPS is the alternative often preferred. Unfortunately, always-on HTTPS can be difficult to deploy, particularly in large web applications not originally designed for such a requirement. Always-on HTTPS not only affects performance (e.g., additional cryptographic overhead, web-caching mechanisms do not work with HTTPS) but also impacts existing functionality (e.g., virtual hosting, applications [80], network content filtering [155]). In addition, even with always-on HTTPS, authentication cookies can be exposed accidentally [74, 43, 120] or stolen by attacking HTTPS [154, 46, 37, 165]. Moreover, HTTPS only protects cookies on the network. An adversary can also steal cookies from the user's computer through many different attacks (e.g., cross-site scripting [103], cross-site tracing [87] and related-domain [25] attacks).

In summary, the simple design of cookies makes them vulnerable to session hijacking. Although authentication cookies are a shared secret between the browser and the web application, they are treated as standard cookies and can be easily disclosed. Consequently, additional protection mechanisms are required to safeguard authentication cookies while traveling on the network and while stored in the browser. This additional protection adds complexity to the security architecture of web applications.

6.2 One-Time Cookies: A Robust and Stateless Session Authentication Protocol

We propose an alternative mechanism to replace cookies as session authentication tokens. Our solution, One-Time Cookies (OTC), provides robust defense against session hijacking while complying with the requirements of highly distributed applications. In particular, OTC separates session authentication from other session management tasks.

6.2.1 Threat model

Our scenario consists of a highly-distributed web 2.0 application, as described in Section 2.3.1. In such scenario, state synchronization among the web application's servers is expensive due to the high network latencies. We assume that all the servers share a long-term secret key that is changed with low frequency (e.g., monthly). This key is equivalent to the key used to generate and verify authentication cookies.

The adversary's goal is to take control of users' sessions with the web application. OTC assumes two types of polynomial-time adversaries (PPT): passive and active. A *passive adversary* has access to all the information exchanged between the browser and the web application. She can access this information directly from the network (online) or from network logs (offline). Based on this information, the passive adversary will try to fabricate or reuse authentication tokens to hijack a user's session. An *active adversary* has the same access to information as the passive one, but in addition, this adversary can actively modify the requests and responses exchanged between the browser and the web application. For example, the active adversary can modify, create and prevent messages from reaching their destination. In addition, an active adversary can execute application level attacks against the browser and the web application, including cross-site scripting (XSS), cross-site tracing (XST) and session fixation attacks. An active adversary can also try phishing attacks to steal OTC tokens or try to steal the OTC's persistent storage file from the user's computer. We do not consider attacks where the adversary takes control of the user's browser or OS (e.g., by exploiting a buffer overflow or through malware) or attacks that compromise the web application infrastructure. Moreover, OTC does not defend against denial of service attacks.

OTC relies on HTTPS to protect the setup of its credentials during the user login. Therefore, OTC assumes that HTTPS is established correctly and in a secure way. We do not consider attacks that break the confidentiality guarantees offered by

HTTPS during user login. If such attacks were possible, the adversary could also steal the user's password – a more valuable credential. However, we do consider attacks against HTTPS connections established after user login.

Based on this threat model, OTC should provide the following security guarantees. First, OTC should offer authenticity and integrity protection to each request sent by the browser to the web application (i.e., user authentication). For this purpose, OTC should rely on a Message Authenticated Code (MAC) primitive that meets the standard notion of unforgeability under chosen-message attack (UF-CMA [77]). Second, OTC tokens should be inherently resistant to session hijacking and replay attacks; additional security mechanisms should not be required to protect requests against such attacks. Third, OTC should not reveal sensitive information (e.g., encryption keys, usernames). For this purpose, OTC should rely on a secure, non-malleable symmetric encryption scheme that meets the standard notion of indistinguishability under chosen plaintext attack (IND-CPA [78]). Finally, OTC credentials should be securely stored and manipulated in the browser.

6.2.2 Desired Protocol Properties

We identified properties required to achieve a robust and practical alternative to authentication cookies. We then used these properties to design OTC:

- *Session Integrity*: the proposed mechanism should provide robust client-side session authentication and it should be inherently secure against session hijacking (i.e., no additional protection mechanisms should be required).

- *Statelessness*: the proposed mechanism should not require additional state in the web application for request verification. In other words, it should not be different from authentication cookies in terms of server state requirements. As described in Section 2.3.1, this property is critical for highly distributed web applications.

- *Robustness*: the proposed mechanism should generate authentication tokens with strong confidentiality and integrity guarantees. In particular, authentication tokens should not leak information that compromises the security of the web application, should be resistant to cryptanalysis attacks (e.g., volume attacks) and should be tamper-evident.

- *Performance and Scalability*: the proposed mechanism should be as efficient and scalable as authentication cookies. Web application's performance and scalability should not be affected.

- *Secure storage*: the proposed mechanism should store authentication credentials securely in the browser. In particular, authentication credentials should be isolated from other browser components and functionality. For example, credentials should have similar protection as passwords and private keys. Any persistent storage should be protected with encryption.

- *Deployability*: the proposed mechanism should require minimal changes in the browser and the web application. No additional hardware or software should be required. In addition, it should be easy to configure in both the browser and the web application.

- *Usability*: the proposed mechanism should provide a similar user experience to cookies. No additional user interaction should be required. In general, the user experience should not change after upgrading from authentication cookies to OTC.

- *Concurrency*: the proposed mechanism should work with web applications that have high request concurrency (e.g., AJAX). Thus, authentication tokens should be independent of each other (i.e., avoid serialization).

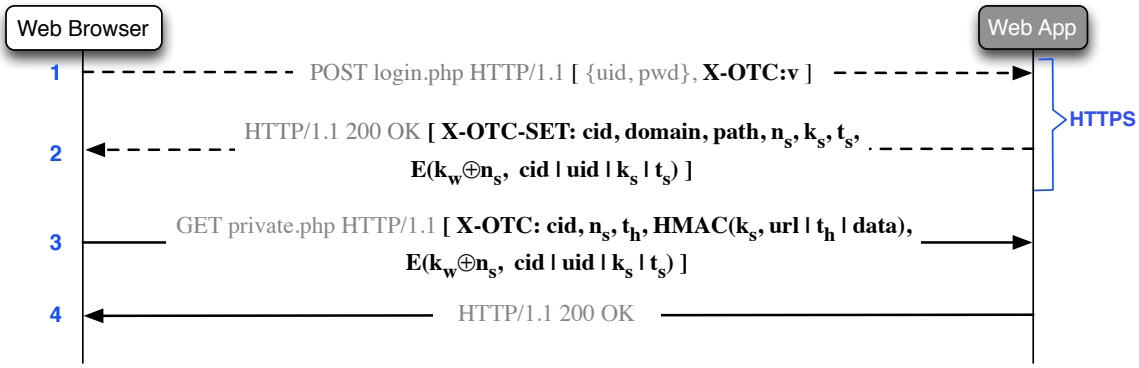
- *Browser support*: The proposed mechanism should be implemented as part of the browser (core component or extension) to provide adequate security and functionality guarantees. This property is important because the mechanism requires access to every HTTP request the browser sends to the web application.

6.2.3 Protocol Description

OTC creates a unique token per request. Each token is bound to a particular request by using a session secret; thus, a token cannot be reused for different requests. In addition, OTC borrows the concept of tickets from Kerberos [142] to store the state information required to validate the token. Each ticket is encrypted with a long-term key shared among all the web application's servers (k_w).[1] Hence, only the web application's servers can access the information stored in the ticket. The user never has access to the contents of this ticket. We define *credentials* as the values stored in the browser and *tokens* as the values attached to each request. OTC tokens are created based on the OTC credentials stored in the browser.

Figure 34 shows how OTC credentials are established and used. During user login (message 1), the browser sends the user's ID (uid) and password (pwd) to the web application. In addition, the browser includes a special HTTP header field: *X-OTC*. This header field indicates that the browser supports OTC session authentication and the OTC protocol version (v). After successful user authentication, the web application checks if the *X-OTC* header field is present in the request. If the field is present, the web application generates OTC credentials for the newly created session. The OTC credentials consist of the credentials' ID (cid), credentials' scope ($domain$ and $path$), a session nonce (n_s), a session key (k_s), a session expiration time (t_s) and a session ticket ($E(k_w \oplus n_s, cid|uid|k_s|t_s)$). The cid, $domain$ and $path$ parameters are used in scenarios where the web application requires more than one set of credentials

[1] Recall that distributed web servers already often share a single key for validating traditional authentication cookies, so we are not requiring any additional state.



uid, pwd	:	user ID and password	cid	:	OTC credential's ID
k_s, n_s	:	session key, session nonce	*domain, path*	:	OTC credential's scope
k_w	:	web application long-term key	$\mathrm{E}(k, x)$	:	symmetric encryption of x with key k
t_s, t_h	:	session and token expiration times	$\mathrm{HMAC}(x)$	:	cryptographic hash-based message authentication code of x
url	:	url of the requested resource			
$data$	:	POST form data	*X-OTC*	:	HTTP header fields for exchanging OTC protocol information
v	:	OTC protocol version			

Figure 34: Flow diagram of a web session using OTC. Messages 1 and 2 represent the user login transaction and require HTTPS protection. After user login, HTTPS is optional; each browser request includes a unique OTC token (message 3) to authenticate the request.

per user. For example, the web application may require one set of OTC credentials for basic operations and another set for administrative operations (see Section 6.4 for an example). If the *X-OTC* header field is not present in the browser request, the web application could switch to standard authentication cookies or halt the communication and notify the user that OTC support is mandatory.

The web application sends the OTC credentials to the browser (message 2) using a special HTTP header field: *X-OTC-SET*. The credentials are sent over the same HTTPS channel used to protect the user's password. Once received, the browser stores the credentials in protected storage, isolated from other browser components.

On every request that matches the credential's scope, the browser attaches an OTC token using the *X-OTC* header field (message 3). The OTC token consists of the credential's ID (cid), the session nonce (n_s), the token's expiration time (t_h),

a hash-based message authentication code (HMAC($k_s, url|t_h|data$)) and the corresponding session ticket ($E(k_w \oplus n_s, cid|uid|k_s|t_s)$). The HMAC computation includes the request's URL (url), the token's expiration time (t_h) and any web form information ($data$) included in POST requests (GET requests' parameters are included in the URL). The OTC token is stateless; *the ticket contains all the information required by the web application to validate the HMAC* (statelessness property, Section 6.2.2). In addition, OTC-tokens are self-contained; thus, they can be verified independently. This property guarantees that OTC tokens can be used in web applications with high concurrency (e.g., AJAX) (concurrency property, Section 6.2.2).

After receiving the request, the web application validates it using the attached OTC token. First, the web application verifies that the token has not expired (i.e., checks t_h). Then, it uses the long-term key (k_w) and the session nonce (n_s) to decrypt the session ticket. If the decryption is successful, the web application validates that the ticket has not expired (i.e., checks t_s) and that the credentials' ID (cid) and user's ID (uid) belong to the current session. Next, the web application computes a new HMAC using the session key k_s and the information in the request (*url*, *data*). It then compares the newly computed HMAC with the HMAC included in the OTC token. If the values match, the request is accepted and the web application returns the requested resource with a *200 OK* HTTP status code (message 4). If the HMAC values do not match or if any of the previous checks fail, the request is denied and the web application redirects the browser to the login page.

The session continues until the session ticket expires (based on t_s) or the user explicitly logs off. To log off, the browsers send a request to the web application with its corresponding OTC token (not shown in Figure 34). The web application verifies the token and sends back a new *X-OTC-SET* header that only includes an HMAC of the value zero (0) using the session key ($HMAC(k_s, 0)$). This HMAC indicates the browser to delete the OTC credentials for this domain (browser enforced policy). By

including this HMAC value, OTC prevents the arbitrary deletion or modification of the OTC credentials via spoofed server responses.

6.3 OTC Security Analysis

6.3.1 Informal Analysis

To provide authenticity and integrity protection to user's requests, a browser supporting OTC signs each request with the session secret (k_s), instead of just attaching it to the requests (as cookies do). Thus, the browser never sends the session secret over the network, reducing its exposure to adversaries. The session secret is generated by the web application's server after successfully validating the identity of the user during the login process. The session secret has long entropy (e.g., 128 random bits) to defend against offline brute force attacks. The server then sends the secret to the browser across the network in response to the login request (message 2 in Figure 34). Therefore, the session secret is protected by the SSL/TLS connection established to protect the user login process. As stated before, OTC assumes that this SSL/TLS connection is correctly established during user login, otherwise an adversary could capture the user's password.

User's requests are signed using a HMAC function [18, 112] and the session secret. As specified in Section 6.2.1, HMAC is UF-CMA secure (HMAC was proven to be a PRF [16] and any PRF MAC is also UF-CMA [17]). This provides formal guarantees that an adversary with reasonable resources will not be able to fabricate or modify OTC tokens without knowledge of the session secret. In is important to note that, while hash functions such as MD5 and SHA-1 are not longer considered secure due to reported collision attacks [195, 194], they can still be used in HMAC functions because HMAC does not require collision resistance for its formal security proof [16, 190]. However, it is recommended that a more robust underlying hash function such as SHA-256 be used in new protocols, as MD5 and SHA-1 support will decrease.

The inclusion of the token's expiration time (t_h) guarantees that each HMAC value is unique, even for identical requests. Thus, the HMAC makes each OTC token unique and ties each token to a particular request. As a result, an adversary (active or passive) will not be able to reuse captured OTC tokens for arbitrary requests; thus, preventing session hijacking attacks (session integrity property, Section 6.2.2).

An active adversary could still try to resend a previously observed request (i.e., replay attack). However, the adversary is limited to replay exactly the same request; she cannot modify the request's payload because it is protected by the HMAC. To make this attack even more difficult, OTC tokens also include an expiration time (t_h). The token's expiration time should have a shorter duration than the session expiration time (t_s). For example, $t_s = 1$ hour and $t_h = 30$ seconds. This approach significantly reduces the window of opportunity for a replay attack. To avoid time synchronization problems, both t_s and t_h should be computed based on the web application's clock.

OTC tokens include a encrypted ticket with the information that a server requires (e.g., session secret, username) to verify the authenticity and integrity of the user request; thus, avoiding additional state in the server. This information is encrypted using a symmetric encryption scheme based on AES as block cipher and Cipher-block chaining (CBC) or Counter (CTR) encryption modes with random initialization vectors (IV). Both encryption schemes are IND-CPA secure; thus, providing strong confidentiality guarantees for the data included in the session ticket. In addition, to increase the difficulty of cryptanalysis attacks, the session tickets are encrypted with a salted version of the long term key k_w (i.e., $k_w \oplus n_s$). Thus, by using the session nonce n_s as a salt for k_w, each session ticket is always encrypted with a different (but related) key.

To hijack a session using OTC, an adversary needs to learn k_w or k_s. We assume that k_w is securely protected by the web application. Thus, it is more likely that an adversary will try to obtain k_s by stealing the OTC credentials from the user's

browser - the only place where k_s is stored. However, OTC credentials are isolated from other browser components by default. In addition, OTC's persistent storage is protected by encryption. Therefore, none of the known cookie-theft attacks are likely to succeed in stealing OTC credentials (secure storage and browser support properties, Section 6.2.2)

In contrast to authentication cookies, OTC requires a signed response message from the server to delete or modify existing OTC credentials in the browser. After a log off request, the server responds with a signed value using the existing session key (k_s). This approach prevents session fixation attacks [111] and "protected" cookie clobbering [206] that take advantage of the fact that cookies can be overwritten by spoofed server responses or malicious JavaScript code.

Compared to cookies, OTC requires a simpler security configuration. OTC does not require additional mechanisms to protect against session hijacking. For example, OTC does not require the *httponly* and *secure* flags, which are often misunderstood or ignored by web developers [208]. Also, OTC does not require always-on HTTPS to prevent session hijacking; thus, providing an alternative to web applications that cannot deploy always-on HTTPS.

Table 5 shows a list of the main threats affecting authentication cookies and if they apply to OTC. Except for denial of service attacks, network attacks do not affect OTC because the browser never sends the OTC session secret across the network. In the browser, OTC is resilient to most attacks affecting cookies because OTC credentials are securely stored and managed in the browser by default. Only attacks where the adversary takes control of the browser (e.g., CSRF, malware) can affect OTC. Therefore, *OTC significantly reduces the attack surface affecting session authentication on web applications.*

Table 5: Main threats affecting authentication cookies. OTC is robust against most of these threats; thus, it effectively reduces the attack surface affecting session authentication based on cookies and simplifies the security architecture of the web application. (Note: x = affected by the threat, - = not affected by the threat).

Threats on the network	**Cookies**	**OTC**
Disclosure due to use of unencrypted HTTP	x	-
Disclosure due to configuration errors/software bugs [74, 43, 120, 81]	x	-
SSL splitting attacks [128, 154]	x	-
SSL renegotiation attacks [46]	x	-
SSL BEAST attacks [165]	x	-
Denial of service attacks	x	x
Threats on the browser and web application	**Cookies**	**OTC**
Cross-Site Scripting (XSS) attacks [186, 103]	x	-
Cross-Site Tracing (XST) attacks [87]	x	-
Cross-Site Request Forgery (CSRF)	x	x
Related-domain attacks [25]	x	-
Clickjacking attacks [156]	x	-
Session fixation attacks [111]	x	-
"Protected" cookie clobbering [206]	x	-
Weak token generation [73]	x	-
Cookie-stealing malware [200]	x	-
Malware controlling the browser	x	x
Social engineering attacks	x	x

6.3.2 ProVerif Analysis

OTC tokens do not leak information that could allow an attacker to learn the session key k_s or the web application's long-term key k_w. To verify this property more formally, we used ProVerif [21, 22], a tool for automatically analyzing the security of cryptographic protocols in the formal adversarial model (i.e. Dolev-Yao model [54]). For this purpose, we modeled the OTC protocol (Figure 34) using *pi calculus*. Using this model and ProVerif, we successfully proved the following OTC's security properties:

- *Secrecy of* k_s: the value k_s is only known to the browser and the web application.
- *Secrecy of* k_w: the value of k_w is only known to the web application.

- *Authentication of the browser to web application*: if the web application reaches the end of the protocol and believes it has shared the k_s with the browser, then the browser was indeed its interlocutor and it has shared k_s.

To test secrecy, ProVerif verified the reachability properties of k_s and k_w based on our model. To test authentication, ProVerif verified correspondence assertions between the two events: when the browser accepted k_s and when the web application finished validating an OTC token. As stated in Section 6.2.1, the symmetric encryption algorithm is assumed to be indistinguishable under chosen plaintext attacks (IND-CPA), while the HMAC scheme is assumed to be unforgeable under chosen message attacks (UF-CMA). The OTC model in pi calculus and the output results from ProVerif are shown in the Appendix A.

These results confirm that, in order to break the secrecy and authentication properties of OTC, an adversary will have to break the security of the underlying cryptographic components: symmetric encryption, HMAC and cryptographic hash functions. Therefore, by observing and/or modifying the communication between the browser and the web application, the adversary gains little advantage against OTC (robustness property, Section 6.2.2).

6.4 Experimental Evaluation

6.4.1 OTC Implementation

We implemented OTC's browser and web application components for our experimental evaluation. In the web application, we added OTC support to WordPress v.3.2.1 [198], one of the most popular open-source web content management system on the Internet. In addition, we configured WordPress with the BuddyPress plugin v.1.5.1 [30] to add more Web 2.0 and social networking functionalities to WordPress. OTC was implemented as a WordPress plugin, requiring less than 200 lines of PHP code. This code replaces the creation and verification functions of authentication

Table 6: Example of authentication cookies and OTC credentials for WordPress. Cookies are setup in the browser with the *Set-Cookie* HTTP header field while OTC credentials are setup with the *X-OTC-SET* header field. On each request that requires authentication, the browser attaches the cookies using the *Cookie* HTTP header field or attaches an OTC token using the *X-OTC* HTTP header field. WordPress uses 3 authentication cookies by default.

WordPress Authentication Cookies
Set-Cookie: wordpress_sec_6e7a6b34f1dd07c511f0105e2f4708a8=admin%7C1320449253%7Cf109f3bb1777a7ca8594286d18e68096; path=/wordpress/wp-content/plugins; secure; httponly **Set-Cookie:** wordpress_sec_6e7a6b34f1dd07c511f0105e2f4708a8=admin%7C1320449253%7Cf109f3bb1777a7ca8594286d18e68096; path=/wordpress/wp-admin; secure; httponly **Set-Cookie:** wordpress_logged_in_6e7a6b34f1dd07c511f0105e2f4708a8=admin%7C1320449253%7C5623496b9ed718a32810ffd056e0d7e8; path=/wordpress/; httponly **Cookie:** wordpress_sec_6e7a6b34f1dd07c511f0105e2f4708a8=admin%7C1320449253%7Cf109f3bb1777a7ca8594286d18e68096; wordpress_logged_in_6e7a6b34f1dd07c511f0105e2f4708a8=admin%7C1320449253%7C5623496b9ed718a32810ffd056e0d7e8;
WordPress OTC Credentials and Token
X-OTC-SET: otc;.myapp.org;/;I9LBXQvMjty1XpcCEl/cvw==;m2jS5QpaaBrfA9iaV8Hqyg==;1320280134;D0ue2+HIm0sFglMJDhtyOAK614mkTWEtC/angk2nQ9acG/AdeKaaF+Z+x1LZn+OU **X-OTC:** otc;I9LBXQvMjty1XpcCEl/cvw==;1320276630;w3rzD/lrPgtG4cv0EQrplg==;D0ue2+HIm0sFglMJDhtyOAK614mkTWEtC/angk2nQ9acG/AdeKaaF+Z+x1LZn+OU

cookies with equivalent OTC functions. The OTC plugin can be installed using the standard WordPress administrative interface in less than 5 minutes (deployability property, Section 6.2.2).

In the browser, OTC was implemented as an extension for Firefox v.7.0.1 and Firefox for mobile (Fennec) v.4.03b (browser support property, Section 6.2.2). The OTC browser extension required approximately 300 lines of JavaScript code. This extension can be installed using Firefox add-ons interface in less than 5 minutes (deployability property, Section 6.2.2). Both, OTC WordPress plugin and Firefox extensions are currently available for evaluation at `http://www.cc.gatech.edu/~idacosta/otc/`. We ultimately envision OTC as being included with core browser functionality, so that all users would benefit without having to install an extension.

Table 6 shows an example of the WordPress authentication cookies and the equivalent OTC credentials based on our implementation. By default, WordPress requires three authentication cookies: two for accessing administrative operations (e.g., changing password) and one for general operations (e.g., posting a new message). The two first cookies, used for administrative tasks, have different scopes and both have the *secure* and *httponly* flags enabled. The purpose of these cookies is to limit the impact of a session hijacking attack. In contrast, OTC only requires a single set of credentials and a single token to authenticate the request because it is inherently robust against session hijacking. In other words, OTC offers simpler but stronger session integrity protection than cookies. As cookies, OTC also allows multiple sets of credentials by using the scope parameters (i.e., domain and path); however, in most scenarios, a single set of credentials should suffice.

As Table 6 shows, OTC uses Base 64 encoding for its credentials and tokens. For a more direct comparison with WordPress cookies, OTC uses an HMAC based on the MD5 [2] cryptographic hash function (HMAC-MD5) and AES with 128 bit keys (AES-128) as a symmetric block cipher and CBC with random IV as the encryption mode. However, it is easy to configure OTC with larger keys or more robust algorithms. Note that symmetric encryption operations are performed by the web application only; the browser does not need to encrypt or decrypt information (i.e., the session ticket).

6.4.2 Evaluation and Results

The main goal of our experimental evaluation is to characterize and compare the performance of OTC and authentication cookies. First, we measured the delay added by single OTC and cookie operations. For this purpose, we used code instrumentation in WordPress and Firefox (desktop and mobile). Second, we characterized the

[2] Collision attacks against MD5 [195] do not affect HMAC security [190]. However, it is recommended to use a more robust hash function such as SHA-256 as MD5 support will decrease.

system-level impact on performance of OTC and cookies on WordPress. We focused on metrics such as page load times, maximum throughput and CPU utilization. All the experiments were executed at least 20 times to ensure the soundness of the results. In addition, mean values and 95% confidence intervals are reported for all the experiments. Finally, we also ran informal experiments to evaluate the usability and compatibility of OTC with WordPress.

6.4.2.1 Microbenchmarks

For a direct comparison with WordPress's cookies, OTC was configured to use HMAC-MD5 and AES-128. In our experiments we used a laptop (MacBook Pro with dual core 2.53 GHz processor, 4GB of memory and Mac OS X 10.6) and a smartphone (Google Nexus One with 1 GHz processor, 512 MB of memory and Android 2.3.6) as our clients. WordPress was installed in an Ubuntu v.8.04 (Linux Kernel 2.6.24) server with 2 Quad-Core 2.00 GHz processors, 16 GB of memory and Gigabit Ethernet cards. The server was also configured with WordPress's supporting software: Apache v.2.2, MySQL v.5.0 and PHP v.5.3. All the server software used a default configuration (i.e., no performance optimizations).

We first measured the time required to generate authentication cookies and OTC credentials and the time required to validate cookies and OTC tokens. Table 10 shows the average results and confidence intervals. For both generation and validation, OTC operations required more time than cookies; however, this difference is negligible when compared to other WordPress operations (performance and scalability properties, Section 6.2.2). For example, loading a WordPress page typically requires hundreds of milliseconds. OTC required 0.2060 ms and 0.3990 ms more than cookies for generating credentials and validating tokens, respectively. These differences are expected because OTC uses symmetric encryption operations and extra validation steps in addition to those used by cookies.

Table 7: WordPress generation and verification times for cookies and OTC. The additional delay added by OTC operations is small (< 1 ms) and negligible when compared to other web application's operations. Note: c.i. = confidence intervals.

Protocol	Generation (ms)	Verification (ms)
Cookies (95% c.i)	0.1610 (±0.0020)	1.6060 (±0.4500)
OTC (95% c.i.)	0.3670 (±0.0120)	2.0050 (±0.0610)

Table 8: Time to generate tokens in the browser. The overhead added is small and unlikely to affect the user experience

Device	OTC Token Generation (ms)
Laptop (95% c.i.)	0.1335 (±0.0034)
Smartphone (95% c.i.)	2.3945 (±0.0974)

On the browser side, we measured the time that OTC requires to generate and attach an authentication token to each request. Table 9 shows the average results and confidence intervals for the laptop and for the smartphone. The results show that the overhead added by OTC to Firefox (desktop and mobile) is also small. For instance, the average network jitter in the US[3] is 0.67 ms, around 4 times the OTC overhead in the desktop browser and 33.41% of the OTC overhead in the mobile browser. Therefore, such delays are unlikely to affect the user experience.

6.4.2.2 Macrobenchmarks

Our first macrobenchmark experiment consisted of measuring the overall latency added by OTC to WordPress's responses. For this purpose, in the browser we measured the time required to load the home page from WordPress for the following configurations: cookies with HTTP, cookies with HTTPS, OTC with HTTP and OTC with HTTPS. We measured this time only for new TCP or SSL/TLS connections (i.e., no channel reuse) using Firebug [69], a Firefox extension for web development. The WordPress home page requires 14 requests for resources (e.g., images, css files) and has a size of 210.4 KB. In addition, we added a latency of 50 ms between the

[3] `http://ipnetwork.bgtmo.ip.att.net/pws/averages.html`

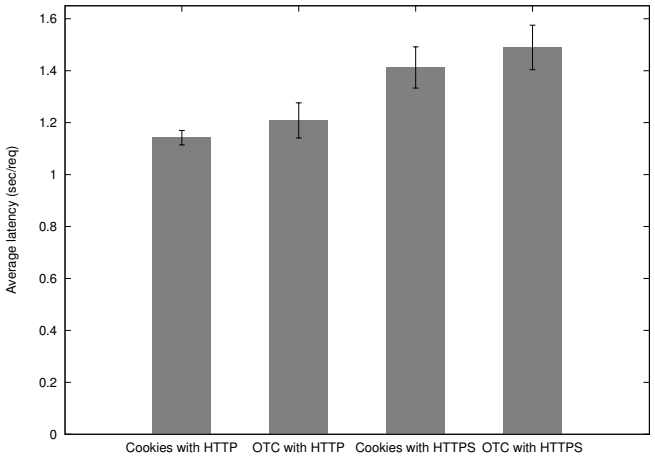
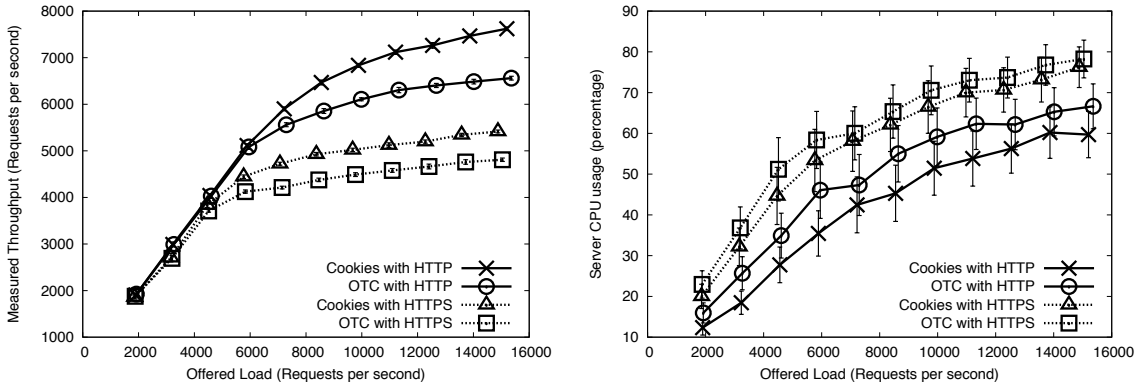


Figure 35: Average user experienced latency per request for cookies with HTTP, cookies with HTTPS, OTC with HTTP and OTC with HTTPS. When compared to cookies, the delay introduced by OTC is small and unlikely to be noticed by the user.

browser and the web application to simulate a more realistic Internet round trip time.

Figure 35 shows the results of this experiment. For HTTP, the WordPress page required approximately 1.14 sec and 1.21 sec to load for cookies and OTC, respectively. For HTTPS, it required approximately 1.41 sec for cookies and 1.49 sec for OTC. Thus, the additional latency introduced by OTC is around 70 ms for HTTP and 80 ms for HTTPS. These values represent to the total time to load the WordPress page, which requires 14 requests. Therefore, the mean latency added by OTC to each request is approximately 5.00 ms and 5.71 ms for HTTP and HTTPS, respectively. Taking into account the margin of error of this experiment, these values resemble the ones measured in our microbenchmarks plus a small amount of network jitter. Therefore, these results confirm that the latency added by OTC to the page load time is negligible (performance and scalability properties, Section 6.2.2).

Our second macrobenchmark experiment measured OTC's overall impact on the maximum throughput that the web application can support. For this purpose, we characterized the maximum throughput (requests per second) for the four configurations described in the previous experiment. For a more realistic comparison of HTTP and HTTPS configurations, we focused on measuring performance during



(a) Web server throughput

(b) Web server CPU utilization

Figure 36: Web server throughput and CPU utilization for cookies with HTTP, cookies with HTTPS, OTC with HTTP and OTC with HTTPS. These tests used a simple PHP page (14 KB) because WordPress throughput was too low (< 100 req/sec) to see performance differences among the configurations evaluated. Cookies and OTC allow similar performance in the web server for practical throughput values.

HTTPS steady state, avoiding the costs of HTTPS connection setup (most expensive SSL/TLS operation [41]). Therefore, in these experiments we used a small and constant number of connections while increasing the number of requests made over these connections (as opposed to increasing the number of connections). In other words, we simulated the load generated by users that already logged in to the web application.

To generate the traffic load, we used *httperf 0.9* [137], a tool for measuring the performance of web servers. A total of three httperf instances (one per server) are used to generate the test loads. We wrote our own custom script to automate execution and data collection of the benchmarking experiments. These servers had a similar hardware configuration to the WordPress server. In addition, our testbed used a dedicated Gigabit Ethernet switch.

In our experiment, instead of requesting a WordPress page, the performance tool requested a small PHP page (14 Kbytes) that contained some blocks of text and OTC and cookie verification support. The reason for this setup is that WordPress pages

are more complex and slower to load, resulting in a much lower throughput (< 100 requests/sec). This low throughput did not allow us to measure the differences among the configurations. In short, WordPress pages become an earlier bottleneck for server performance instead of cookies or OTC.

The results for request throughput are shown in Figure 36a. As expected, the configurations using HTTP performed better than the configurations using HTTPS. When HTTP is used, the web application supported a maximum throughput of approximately 7,500 requests/sec for cookies and 6,500 request/sec for OTC. These results represent a reduction in request throughput of approximately 13.33% when OTC is used instead of cookies. When HTTPS is used, the web application supported a maximum throughput of approximately 5,400 requests/sec for cookies and 4,900 requests/sec for OTC. These results represent a reduction in request throughput of approximately 9.26% when OTC is used instead of cookies (the smaller difference is due to the overhead added by HTTPS). Figure 36b shows the web server CPU utilization during the experiments for each configuration. When HTTP is used, OTC requires around 10% more CPU time than cookies due to the symmetric encryption operations. When HTTPS is used, OTC requires around 5% more CPU time than cookies. This difference is almost negligible, as Figure 36b shows, because of the HTTPS overhead. As in our previous experiment, these results show that OTC introduces a small performance overhead to the web application. While a 13.33% reduction in throughput is not negligible, note that these measurements were taken using a lightweight PHP page. Our WordPress implementation cannot support more than hundred requests per second; therefore, the overhead added by OTC is insignificant when compared to WordPress's own overhead (performance and scalability properties, Section 6.2.2). In addition, these results show that OTC can be used with both HTTP and HTTPS configurations. Finally, it is important to note that, as a mechanism to prevent session hijacking, HTTPS adds a considerably higher overhead than OTC.

6.4.2.3 Informal Usability and Compatibility Test

We also evaluated OTC's impact on user experience. We conducted a very informal user study where we asked other lab members to use our WordPress setup with both Firefox and Firefox for mobile. None of the participants reported any difference between using cookies or OTC (usability property, Section 6.2.2). In addition, we thoroughly evaluated the functionality of WordPress and BuddyPress to detect any errors or compatibility problems when OTC was used. During this compatibility test, we did not find any problem with Firefox or with WordPress/BuddyPress functionality when OTC was used instead of cookies.

6.5 Discussion

6.5.1 Incrementally Deploying OTC

As our implementation shows, OTC can be easily deployed in today's web applications and browsers. For most web applications, the operations required to support OTC are not different from the ones currently used to support authentication cookies. The use of symmetric encryption by OTC could be considered the main difference. However, it is not uncommon for cookies to use symmetric encryption to protect sensitive user's session information.

The major difference between deploying OTC and cookies is in the browser. With OTC, the browser acquires an active role during session authentication by signing each user request (as opposed to just storing and attaching cookies to requests). Still, the operations required by OTC are already supported by most browsers' APIs (e.g., cryptographic hash operations, secure storage). We expect that OTC support in the browser will follow a similar adoption model as ForceHTTPS [100]: first available as a browser extension and then adopted natively by major browser vendors as an Internet standard (i.e., HSTS). As part of our future work, we plan to collaborate with vendors and groups such as the IETF to propose OTC as an Internet standard.

Most web applications can follow an incremental approach to deploy OTC. For example, web applications can enable OTC while still supporting authentication cookies. Initially, a small group of users (i.e., beta testers) can evaluate the web application functionality using OTC while standard users continue using authentication cookies. Next, the web application can enable OTC support for all users, indicating how to activate it on browsers (e.g., browser upgrade or through an extension). At this point, both OTC and cookies will be allowed for session authentication; the browser will indicate to the web application the type of protocol preferred. Thus, users that have not updated their browsers, will still be able to access the web application. Note that downgrading attacks are unlikely when OTC and cookies are both enabled because OTC support is announced over HTTPS during user login (see Section 6.2.3). After certain threshold is reached (e.g., percentage of users supporting OTC), the web application can deprecate the use of cookies for session authentication and rely on OTC only. Web applications with high security requirements (e.g., online banking) can combine OTC with always-on HTTPS to add another layer of security to their systems. This type of applications can follow a more aggressive approach and enable OTC directly as the only session authentication mechanism (i.e., no transitional period). Thus, users will be required to update their browsers to use the application. Finally, OTC will not replace cookies for other session management tasks (e.g., shopping card, user preferences); cookies will still be needed for such functionality.

6.5.2 Extending OTC Integrity Protection

In addition to protecting the integrity of user's requests, OTC could also protect the integrity of the web application's responses. This approach can provide lightweight integrity protection in scenarios where it is difficult to deploy always-on HTTPS. For this purpose, the session key k_s could be used by the web application to sign the resources sent to the browsers (e.g., HTML code, JavaScript code, CSS code).

Because the session key is included in each request, the web application can readily use it to sign the corresponding response (i.e., no delay is introduced by key retrieval operations). The OTC browser component will require only minor changes to support verification of web responses.

By signing web application's responses, OTC could detect in-flight page modifications (e.g., ISPs injecting adds, adversaries injecting malicious code). Web Tripwires [157] was proposed to detect such activity; however, it is not robust against some active adversaries because it does not rely on a shared secret. OTC could be combined with mechanisms such as Web Tripwires to provide a more robust integrity mechanism for web application's responses. We plan to explore this approach in future work.

6.5.3 OTC and Multi-Factor Authentication

Cookies are also used as lightweight second-factor authentication tokens. For example, mechanisms such as Yahoo's Sign-In Seal use long-lasting cookies to store a second-factor authentication token in the browser. The advantage of this approach is that the browser does not need modifications. However, some of these mechanisms are not effective against phishing attacks [164]. These cookies can also be stolen from the user's browser. Ben Adida proposed Beamauth [6], a more robust alternative based on specially crafted bookmark instead of a cookie. The use of bookmarks to store second-factor tokens offers better protection against cookie-theft attacks. But, as in the case of cookies, bookmarks were not designed for storing security-related information. Thus, OTC could provide a more robust alternative in this area. For example, a long-lasting OTC credential and its corresponding session key k_s could be used as a second-factor token. Because the browser isolates OTC credentials from other components, k_s offers better security guarantees than cookies or Beamauth as a second-factor authenticator.

6.5.4 OTC in Mobile Devices

Mobile devices such as smartphones and tablets are rapidly becoming the main platforms to consume Internet content. However, due to hardware constraints and lack of security maturity, these devices are more vulnerable to security threats, including session hijacking. Mobile browsers and many mobile applications rely on cookies for session authentication. But, as with desktop computers, cookies can be easily exposed. However, the common wisdom is that smartphones are less vulnerable to session hijacking because they use the mobile operator network for Internet access instead of Wi-Fi. Unfortunately, this scenario is quickly changing. The popularity of mobile devices has created capacity problems for mobile operators, that have been forced to introduce data caps. As a result, smartphone users are increasingly relying on Wi-Fi networks to access the Internet [42]. In addition, mobile operators are also exploring how to reduce the load in their networks by using automatic Wi-Fi offloading mechanisms [62]. This trend significantly increases the risk of mobile devices being targeted with session hijacking attacks and other Internet threats. For this reason, we decided to also implement OTC as part of a mobile browser. As our experimental evaluation shows, OTC is lightweight enough to be used in mobile devices such as smartphones, tablets and other resource-constrained devices.

6.5.5 SessionLock

As mentioned in Section 3.2.1, SessionLock [7] is another proposed technique to prevent session hijacking without requiring always-on HTTPS support. As OTC, SessionLock uses a session secret to sign each request sent to the web application, creating unique authentication tokens per request (prior to SessionLock, Liu et al. [123] and Blundo et al. [24] have also explored this approach). The main novelty of SessionLock is that it uses URL fragment identifiers to store the session secret in the browser and only employs client-side JavaScript to sign requests with this secret. Thus, compared

to OTC, SessionLock offers better deployability because it does not require changes to the browser, only to the web application. However, this improved deployability comes at a cost – reduced reliability and robustness due to the limitations of client-side JavaScript. For example, client-side JavaScript cannot modify browser requests dynamically. Thus, in order to sign every possible request, SessionLock needs to rewrite every link on each web page displayed to the user. This operation can be computationally expensive for complex web pages and will not work with binary objects (e.g., Flash). This technique will also fail if the user types the URL or opens a new browser tab to send a request to the web application. Moreover, SessionLock's session secret can be accidentally leaked by the user (e.g., by sharing a link, bookmarks) or lost during the session. In addition, the use of JavaScript-only makes SessionLock vulnerable to active attacks (e.g., code injection) that could compromise the session secret[4]. These problems could be avoided if SessionLock is implemented on the browser (as OTC) instead of relying on client-side JavaScript only. Still, SessionLock requires additional state on the web application for the session secret; thus, it is not appropriate for highly distributed web applications (see Section 2.3.1). From the efficiency perspective, both SessionLock and OTC have similar performance profiles as they execute comparable operations. OTC requires additional computation on the server due to encryption operations associated with session tickets, but the difference should be negligible in most cases. Furthermore, both solutions are transparent to the user. Also, OTC and SessionLock cannot prevent attacks based on CSRF, social engineering or malware and both are susceptible to implementation errors. In summary, while SessionLock allows easier deployment by avoiding browser modifications, its ad hoc techniques are not adequate for complex, highly distributed web applications. As a result, SessionLock has not been deployed in production systems.

[4]In the paper [7], the author described this weakness and stated that SessionLock was designed to prevent session hijacking attacks only.

6.6 Summary

The risks associated with the use of cookies as session authentication tokens have been known for years. More robust alternatives have been proposed to replace authentication cookies; however, they have not been deployed because they fail to meet the requirements of highly distributed Web 2.0 applications. Specifically, most of the proposed alternatives require costly state synchronization across the web application, a serious concern for distributed systems. In this chapter, we presented OTC, a principle-driven secure alternative to authentication cookies. OTC is not only resistant to session hijacking, but also maintains the simplicity and performance benefits of cookies. More critically, OTC addresses the shortcomings of previously proposed solutions by removing the need for state in the web application. Moreover, OTC offers another security layer to web applications that already support always-on HTTPS by reducing the threats associated with cookies; OTC and always-on HTTPS are complementary mechanisms. We developed OTC for the popular WordPress application and demonstrated that OTC has similar performance to traditional cookies.

CHAPTER VII

DVCERT: ROBUST SERVER AUTHENTICATION FOR SSL/TLS WITHOUT THIRD-PARTIES

The Secure Sockets Layer (SSL) protocol and its successor, Transport Layer Security (TLS), have become the de facto means of providing strong cryptographic protection for network traffic. Their near universal integration with web browsers arguably makes them the most visible pieces of security infrastructure for average users. In the previous chapter, we described how SSL/TLS is the recommended mechanism to protect authentication cookies against session hijacking attacks. Our proposed alternative to authentication cookies, OTC, also relies on the security guarantees offered by SSL/TLS, but only during its setup. While vulnerabilities are occasionally found in specific implementations, SSL/TLS are widely viewed as robust means of providing confidentiality, integrity and server authentication. However, these guarantees are built on tenuous assumptions about the ability to authenticate the server-side of a transaction by using digital certificates signed by a *trusted* third-party certification authority (CA).

The security community has long been critical of the Public Key Infrastructure for X.509 (PKIX) and its CA-based trust model [63, 90, 5]. Much of the concern has focused on the role of the CAs and their ability and motivation to not only correctly verify and attest the coupling between an identity and a public key, but also to protect their own resources. Browsers and operating systems determine what CAs users should trust by default (i.e., trust anchors). However, this model has resulted in *hundreds of CAs, all equally trusted and from more than 50 different countries* [160, 58]. Due to this excessive trust, CAs can forge certificates for any

domain that will be accepted as valid by most browsers. Thus, adversaries can obtain forged certificates by coercing or compromising any CA and use them to execute man-in-the-middle (MITM) attacks against SSL/TLS connections. In 2011, the number of reported attacks against CAs increased considerably [158, 107, 76, 85, 109, 56, 131]. In some cases, adversaries were able to forge certificates for important web domains (e.g., google.com, yahoo.com and live.com). Even worse, it has been estimated that a forged certificate was used to intercept close to 300,000 Gmail sessions in Iran [121]. Furthermore, there is evidence that governments and private organizations are using forged certificates as part of their surveillance and censorship efforts [172, 84, 176, 122]. The frequency of these incidents is likely to increase in the future, as more and more web applications rely on SSL/TLS to protect all their communications.

Multiple solutions have been proposed to deal with the threat imposed by forged certificates and MITM attacks (see Section 3.2.2). The most popular approach is the use of additional third-parties to extend or replace the rigid CA trust model (e.g., network notaries [196, 129], public audit logs [61, 118] and secure DNS (DNSSEC) [95]). In this approach, users can select one or more third-parties to vouch for the authenticity of a certificate, improving the chances of detecting a MITM attack. However, depending only on third-parties for certificate validation has several shortcomings such as: significant deployment and operational costs (e.g., additional infrastructure with high availability requirements), more complex trust model for users, privacy concerns and more complex revocation procedures. Therefore, *the inherent complexity and costs associated with third-party solutions have prevented their widespread deployment*. As a result, most users still rely on weak certificate validation checks to detect MITM attacks.

In this chapter, we propose Direct Validation of Certificates (DVCert), an efficient and easy to deploy protocol that provides stronger certificate validation (i.e., server authentication) and effective detection of MITM attacks without using third-parties.

Our mechanism comes from a simple observation – users have already established secrets (e.g., passwords) with their most important web applications. *DVCert allows web applications to use these secrets to directly and securely attest for the authenticity of their certificates without exposing those secrets to offline attacks.* After a single round-trip DVCert transaction, a browser receives the information required to validate all the certificates that could be used during a session with the web application, including certificates from other domains. As a result, to execute a MITM attack, an adversary not only needs to compromise a CA but also each targeted web domain. A DVCert transaction uses a modified Password Authenticated Key Exchange (PAKE) protocol known as PAK [26, 126]. However, we are not simply applying a known protocol; rather, we modified PAK to provide *only* server authentication and integrity protection instead of mutual authentication and generation of encryption keys (i.e., traditional use of PAKE protocols). These changes allow better performance and simplify deployment without affecting PAK's formal security proofs. Our experimental evaluation shows that an optimized DVCert transaction requires little computation time on the server (e.g., < 1 ms) and on the browser. More importantly, DVCert transactions are executed at most once per session; thus, their impact on server performance or user experience is negligible. DVCert's design also provides multiple advantages over third-party solutions: simpler trust model, lower deployment and operational costs (e.g., no additional infrastructure is required) and no privacy risks. Finally, DVCert is a readily available mechanism designed to improve the current CA trust model and be compatible with third-party solutions such as DNSSEC, once these solutions are deployed in the future.

In so doing, we make the following contributions:

- **Designing and implementing an efficient and easy to deploy mechanism to detect MITM attacks against SSL/TLS without third-parties:** We identify key properties required to achieve a robust and practical defense

against MITM attacks. Based on these properties, we develop a protocol that provides more robust certificate validation and detects MITM attacks, even if the adversary uses forged certificates. By allowing web applications to attest directly for their certificates, our mechanism avoids many of the challenges hindering the deployment of third-party solutions. We implemented a proof-of-concept extension for Firefox and Firefox for mobile browsers and a PHP-based server component to demonstrate the deployability of our solution.

- **Conducting an extensive performance analysis in multiple platforms:** We characterize DVCert's performance using our prototype implementation in both desktop and mobile browsers. Our results show that an optimized DVCert transaction requires 0.54 ms of computation time on the server and 12.03 and 97.70 ms on a laptop and on a smartphone respectively. Compared to a naïve implementation, these results represent a 94.96%, 55.07% and 77.82% improvement on the server, laptop and smartphone correspondingly. Moreover, our experimental evaluation demonstrates that DVCert transactions are as efficient as existing server operations (e.g., processing HTTPS requests). Thus, given their low frequency, DVCert transactions are unlikely to degrade server performance or scalability. Furthermore, we apply ProVerif [22, 21] to formally verify DVCert's resilience to offline dictionary attacks.

- **Making our DVCert implementation available to the community:** The DVCert extension for Firefox and Firefox for mobile as well as the server PHP code are available for evaluation at: `http://www.cc.gatech.edu/~idacosta/dvcert/index.html`.

The remainder of this chapter is organized as follows: Section 7.1 offers important background information on MITM attacks and presents our motivation; Section 7.2

provides the design and formal description of DVCert; Section 7.3 presents our security analysis of DVCert; Section 7.4 shows our experimental analysis and results; Section 7.5 offers additional analysis and discussion of our proposed protocol; and Section 7.6 presents a summary of our work.

7.1 Background and Motivation

7.1.1 The SSL/TLS Protocols and Web Applications

The SSL/TLS protocols [72, 53] are the main security mechanisms used to protect the communications between browsers and web applications. By providing a transparent encryption layer, SSL/TLS guarantee the confidentiality and integrity of the data traveling across the Internet. Moreover, SSL/TLS allow browsers to authenticate web application's servers via X.509 digital certificates [4]. A digital certificate binds the server's identity (i.e., domain name) to the server's public key and it is signed by a Certification Authority (CA) trusted by both the server and the browser (Section 2.3.7 describes how a SSL/TLS connection is established and how the server is authenticated). CAs are required because the browser and the server do not share any secrets at the SSL/TLS layer; thus, a trusted third-party is needed to vouch for the authenticity of the server's certificate. Certificates can also be used for user authentication; however, this is not a common practice in Internet scenarios.

Initially, due to performance considerations, most web applications used SSL/TLS only to protect requests carrying private data (e.g., passwords, credit card numbers). However, due to the increasing number of attacks against web sessions (e.g., session hijacking), many applications have been forced to protect all their communications with SSL/TLS. For this reason, it is common that during a session, a browser establishes multiple SSL/TLS connections not only with web application's servers but also with servers from third-party domains (e.g., CDNs and ads networks). Through a short survey from the Alexa Top 20 US sites and popular online banking sites (15

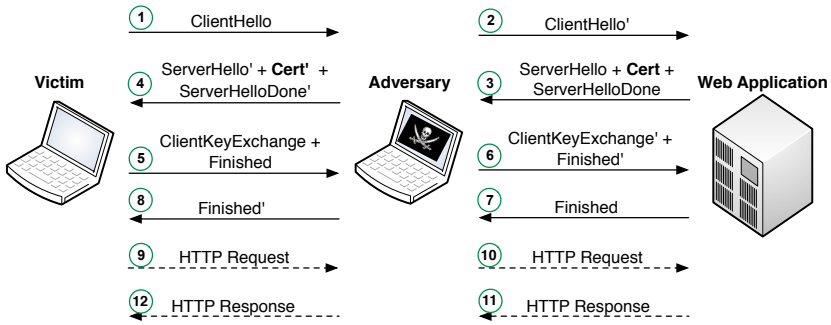


Figure 37: Example of a MITM attack against SSL/TLS. The adversary establishes two SSL/TLS connections: one with the victim and one with the client. However, from the victim's and server's point of view, there is only a single SSL/TLS connection.

in total), we determined that an average of 12 certificates per domain were validated by the browser, with a minimum of 4 and a maximum of 22. Moreover, most sites included at least one certificate from a third-party domain.

7.1.2 MITM Attacks against SSL/TLS

The security guarantees offered by SSL/TLS rely on the correct authentication of the server. All such guarantees are rendered ineffective if an adversary is able to convince users to accept an illegitimately generated certificate, as shown in Figure 37. First, the adversary positions herself in the network path between the victim's computer and the server. When the victim sends a request for establishing a new SSL/TLS connection with the server (message 1), the adversary intercepts and responds to it (message 4) using a forged certificate (Cert'). If the victim accepts this certificate, then she completes the SSL/TLS setup with the adversary (messages 5 and 8), who has, as a result, successfully masqueraded as the server. Simultaneously, the adversary establishes a new SSL/TLS connection with the server (messages 2, 3, 6, and 7). At this point, the adversary has two active SSL/TLS connections: one with the victim and one with the server. However, from the victim's and server's perspectives, there is only one secure connection in place. The adversary can now decrypt, re-encrypt and forward all the messages exchanged between the victim and the server

(messages 9 to 12). As a result, the adversary can access private information (e.g., passwords) or even modify it (e.g., code injection). Finally, notice that the server cannot determine if it is communicating with the victim or the adversary because server and user authentication happen at different independent layers (i.e., weak mutual authentication).

For an adversary, the easiest way to execute a MITM attack is to use self-signed certificates. While such certificates will fail the browser's validation checks, several studies show that average users are likely to ignore browser's warnings [52, 164, 180, 93]. However, as mentioned earlier, a more effective approach is to use a certificates forged by a trusted CA to avoid any browser's warnings. An adversary only needs to deceive, coerce or compromise one of the hundreds trusted CAs to obtain forged certificates for the targeted domains. As many recent incidents show [158, 107, 76, 85, 109, 56, 131], such attacks are becoming increasingly popular. Furthermore, problems with certificate revocation mechanisms (e.g., CRLs, OCSP) [117] provide additional advantage to adversaries.

7.1.3 Problems with Third-Party Solutions

A considerable number of mechanisms have been proposed to improve server-side authentication and protect against MITM attacks (see Section 3.2.2). The most popular approach is the use of additional third-party entities that can also vouch for the authenticity of server certificates. Third-party solutions provide a number of benefits: protection of the first connection to a new domain, scalable attestation of certificates for all public domains and minimal requirements for web applications. Unfortunately, this approach also faces several critical challenges. First, *these mechanisms have significant deployment and operational costs.* The additional infrastructure needed can be expensive to deploy and operate due to requirements such as high-availability, data consistency, performance and security. Even web applications can be affected

by the operational overheads required by these mechanisms. Second, *the resulting trust model is more complex*. The use of multiple trusted entities to choose from can make the trust model more complex to evaluate and understand. Thus, average users are likely to rely on default trust configurations. Moreover, trust is dynamic – a trusted entity today may become an adversary tomorrow. Third, *these mechanisms introduce new privacy risks*. Users' browsing activity is disclosed to third-party entities. Preventing this problem can add complexity to these solutions. Fourth, *certificate revocation procedures become more complex*. The use of multiple entities make revocation more difficult because of the additional overhead required to revoke multiple proofs of authenticity (e.g., signatures). Finally, *captive portals typically interfere with these mechanisms*. In places such as airports and hotels, captive portals can block requests for certificate validation to external entities before user registration. Thus, captive portals need to be modified to allow additional certificate validation mechanisms.

7.2 Direct Validation of SSL/TLS Certificates

We present Direct Validation of SSL/TLS Certificates (DVCert), an efficient and practical mechanism that improves certificate validation and provides stronger protection against MITM attacks. Instead of relying on third-parties, DVCert uses the existing shared secrets between the user and the web application to directly validate server certificates. DVCert overcomes the limitations of third-party solutions while also reducing the risks associated with using low-entropy keys in network protocols.

7.2.1 Scenario and Threat Model

Our scenario assumes a large, highly distributed web application. The application uses SSL/TLS to protect all the communications with its users (i.e., always-on HTTPS). To establish SSL/TLS connections, the application has multiple certificates signed by a trusted CA. In addition, the application's web pages include content from

third-party servers. These servers also communicate using SSL/TLS and have their own valid certificates. We assume that SSL/TLS are correctly configured in the application's servers as well as in the third-party servers. Furthermore, users share a password with the application and use HTML forms for authentication. Instead of plaintext passwords, the application stores password salted hashes using public salt values. Finally, we assume that users follow a robust password policy that is enforced by the application.

We consider a polynomial-time (PPT) adversary that has access to all the communication between the web application and its users. The adversary's goal is to eavesdrop and tamper with this communication by executing MITM attacks against SSL/TLS. To perform such attacks, we assume that it is possible for the adversary to obtain forged certificates for any domain that are signed by some trusted CA. However, the adversary does not have access to users' passwords, password salted hashes or server's private keys. Moreover, this model does not consider attacks against user computers or application servers to obtain such information and attacks that exploit SSL/TLS implementation or configuration errors.

Based on this threat model, DVCert should provide the following security guarantees. First, it should provide authenticity and integrity protection of the application's certificate information sent by the server to the browser. For this purpose, DVCert should rely on the user's password, a PAKE protocol with formal security proof and a collision resistant hash function (e.g., SHA-256). Second, DVCert should allow the browser to perform a more robust server certificate validation and detect MITM attacks based on forge certificates. Third, DVCert should not leak sensitive information such as usernames and passwords. Thus, DVCert should rely on a PAKE protocol to protect users' passwords. Finally, DVCert protocol information store in the server and the browser should not allow adversaries to impersonate users.

7.2.2 Desired Protocol Properties

We identified properties required to achieve an effective and practical defense against MITM attacks. We then used these properties to design DVCert:

1. *Effective detection of MITM attacks*: the proposed mechanism must provide robust server authentication and effective detection of MITM attacks against SSL/TLS, even if illegitimately obtained certificates are used.

2. *Robustness against offline attacks*: the proposed mechanism should not leak information about the user's authentication credentials and must be resilient to offline attacks such as dictionary and cryptanalytic attacks.

3. *Deployability*: the proposed mechanism should not require additional hardware or software, only small changes to the browser and web application. In addition, it should be simple to configure in both the browser and the web application.

4. *Performance*: the proposed mechanism must be efficient. It must not affect the overall performance and scalability of the web application. Moreover, it should not introduce risks of DoS attacks.

5. *Privacy*: the proposed mechanism should not disclose user information to third-parties and adversaries.

6. *Compatibility*: the proposed mechanism must not interfere with existing functionality in the browser and web application. Browsers not supporting the proposed mechanism should still be able to access the web application. Moreover, the proposed mechanism must be compatible with other certificate validation protocols.

7. *Usability*: the proposed mechanism should require minimal user intervention and have minimal impact on user experience.

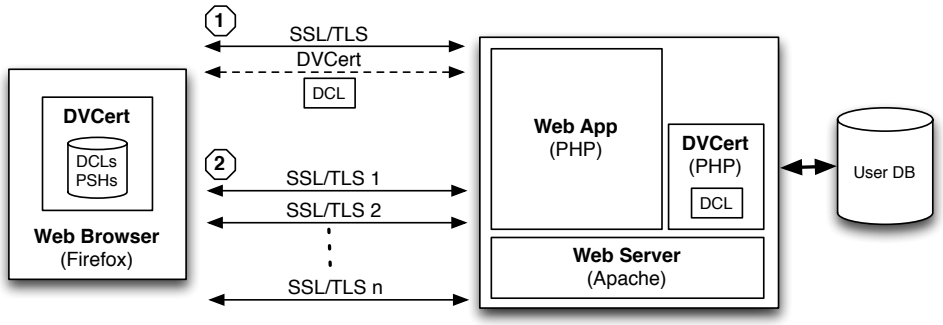


Figure 38: High level overview of the DVCert protocol. First, the browser obtains a fresh DCL (Domain Certificate List) after executing a DVCert transaction over SSL/TLS with the web application (step 1). Second, the browser uses the fresh DCL to validate the certificates used in all the SSL/TLS connections with the web application and associated third-parties (step 2).

8. *Simple trust model*: the proposed mechanism should have an easier to understand trust model in comparison to third-party solutions. Users must not be required to make additional trust assessments.

7.2.3 Protocol Description

MITM attacks against SSL/TLS connections are possible because server certificates are validated using only a single third-party signature and mutual authentication is weak. DVCert addresses these problems by allowing web applications to use already available shared secrets to *vouch directly* for the authenticity of certificates instead of relying only on third-parties. Figure 38 shows a high level description of the DVCert protocol. First, the browser establishes a SSL/TLS connection with the web application and then executes a DVCert transaction based on the user's password and a modified PAKE protocol (step 1). In this transaction, the browser authenticates the web application and receives its latest certificate information. The certificate information is shared using a Domain Certificate List (DCL), a data structure maintained by the web application that contains the *fingerprints*[1] *of all the certificates* that could

[1]A certificate fingerprint is the cryptographic hash of the binary representation (e.g., DER encoding) of the certificate.

be used during a session with the application. The DCL not only includes the fingerprints of the application's certificates but also of third-party's certificates used in the application (e.g., CDNs and ads networks). Second, the browser stores the DCL temporarily and uses it to validate the certificates of each SSL/TLS connection with the application (step 2), including the SSL/TLS channel established in step 1. If a certificate is not found in the DCL, then the corresponding SSL/TLS connection is flagged as untrusted (i.e., probable MITM attack). Once the DCL expires, a new DVCert transaction is executed (step 1) to update it. Finally, to avoid asking for the user's password on each transaction, the browser securely stores the password salted hash (PSH) together with the DCL.

DVCert achieves our goals by building on a significantly modified version of PAK [26, 126, 29, 98]. PAK (and the PAKE family of protocols) is based on the Diffie-Hellman (DH) key exchange and allows the use of low entropy secrets such as passwords to securely establish a session secret (i.e., authenticated Diffie-Hellman). PAK was selected as a starting point for our work because of its formal security proof and its ability to use shorter exponents [127] for better performance when compared to other related PAKE-based protocols. The major difference in our approach is that DVCert uses PAK *only* for server authentication instead of mutual authentication and generation of encryption keys (standard use of PAKE protocols), and include features to protect the integrity of the DCL and distinguish between tampering of the DCL and password errors. In other words, only the browser verifies the session secret established during the transaction. By not providing user authentication, DVCert requires fewer messages and, more importantly, avoids changes to the browser login user interface – a major challenge for the deployment of PAKE protocols in web applications [65]. Hence, DVCert is compatible with current user authentication mechanisms (e.g., HTML form-based authentication).

Figure 39 shows the details of a DVCert transaction (step 1 on Figure 38). First,

Shared information: g, p, $d = domain$, $s = H(u|d)$. Hash functions H, H_1, H_2, H_3, H_4
Information held by Browser: $u = username$, $pw = password$
Information held by Server: $P = H(pw|s)$, DCL = *domain certificate list*

Browser			**Server**
		SSL/TLS setup	
$a \in Z_q$ $P = H(pw\|s)$			
$m_1 = g^a \times H_1(u\|d\|P) (\text{mod } p)$	(1)	$u,\ m_1$ →	$m_1 \text{ mod } p \stackrel{?}{\neq} 0$ $b \in Z_q$ $g^{ab} = (\frac{m_1}{H_1(u\|d\|P)})^b (\text{mod } p)$ $m_2 = g^b \times H_2(u\|d\|P) (\text{mod } p)$ $r = (u\|d\|P\|g^a\|g^b\|g^{ab})$ $h_1 = H_3(r\|H(DCL))$
$g^{ab} = (\frac{m_2}{H_2(u\|d\|P)})^a (\text{mod } p)$ $r = (u\|d\|P\|g^a\|g^b\|g^{ab})$ $h_1 \stackrel{?}{=} H_3(r\|H(DCL))$ $h_2 \stackrel{?}{=} H_4(r)$	(2)	← $m_2,\ h_1,\ h_2,\ DCL$	$h_2 = H_4(r)$

Operations:
$x|y$: concatenation of strings x and y
$H^i(x)$: *i-th* standard cryptographic hash of x
$H_i(x)$: special agreed-on cryptographic hash of x [29, 98]

Figure 39: Detailed description of a DVCert transaction. DVCert uses a modified version of PAK to establish a session secret (g^{ab}) that is used to protect the integrity of the DCL (Domain Certificate List). At the end of the transaction, the server is authenticated and the browser can use the DCL to verify all the certificates used during a session with this domain.

the browser establishes a SSL/TLS connection with the server. This connection is used to protect protocol information (e.g., usernames) from eavesdroppers. Next, the browser generates a random exponent a (browser's DH secret), computes the DH value g^a and uses it and the password salted hash P to compute m_1. If the password salted hash is not available for this domain (e.g., first DVCert transaction with this domain), then the browser prompts the user for her username u and password pw, computes the password salted hash P and stores it in a secure location for future transactions (i.e., the user is prompted only once for her password). Once m_1 has been calculated, the browser sends it and the username u to the server using a special

header field in a HTTP request (message 1) over SSL/TLS. After receiving the DVCert request, the server verifies that $m_1 \neq 0$ to prevent a known attack, uses the username u to retrieve the password salted hash P from the server's database, generates the random exponent b (server's DH secret) and computes the DH value g^b. The server now obtains the browser's DH value g^a from m_1, calculates the session secret g^{ab} and computes m_2 and h_2. In addition, the server uses the latest version of the *DCL* to compute h_1. Next, the server sends m_2, h_1, h_2 and the *DCL* to the browser in the HTTP response (message 2). Then, the browser uses the received values to obtain the server's DH value g^b and to calculate the session secret g^{ab}. Next, the browser uses the session secret g^{ab} and other protocol state information to compute new h_1 and h_2 values. The browser now compares the computed h_1 with the one received from the server. If the values match, then the DVCert transaction was successful. Thus, the DCL file is trusted (i.e., has not been tampered with) and can be used to validate certificates. In addition, the successful verification of h_1 also proves the server's identity. If the h_1 values do not match, then the browser proceeds to verify h_2. If this verification succeeds, then the DCL has been modified and there is a high probability that a MITM is in progress. Therefore, neither the DCL nor any communication with the server can be trusted. The browser displays a warning to the user and halts the communications with the server. If the h_2 values are different, then the transaction could have failed due to a password error (e.g., user typed the wrong password) or a MITM attack. Thus, the browser displays a warning and prompts the user for a new password for a limited number of attempts. If the protocol still fails after several attempts, then the browser halts all communications with the server. In other words, h_2 is used to differentiate between protocol failures due to a MITM attacks or due to password errors.

After a successful DVCert transaction, the browser stores the DCL and the password salted hashes in a secure location isolated from other browser components. The

browser stores one DCL per domain for a limited period of time according to a domain policy (e.g., once per session). Thus, the total number of DVCert requests per user is significantly lower than the total number of SSL/TLS connections. When a SSL/TLS connection is established with a server, the browser checks that the certificate is in the corresponding DCL (step 2 in Figure 38). If the certificate is not in the DCL, then a MITM attacks is likely to be in progress. Thus, the browser displays a warning to the user and halts the communications with the server. Once a DCL expires, the browser sends an automatic request (i.e., no user intervention) for a new DVCert transaction to update the DCL.

Finally, DVCert assumes that PAK constants, the prime number p and the generator g, are publicly known. For example, they can be hardcoded in DVCert's browser and server components. This measure is important to prevent an adversary from sending bogus p and g values and tricking the user into an improper DVCert exchange that could leak password information. Moreover, DVCert assumes that the web application stores password salted hashes ($P = H(pw|s)$) and that salt values (s) are also publicly known. If the salt is not known in advance, the browser can also send an additional request to the server to obtain it.

7.3 Security Analysis

DVCert main's goal is to detect MITM attacks against SSL/TLS. DVCert achieves this by effectively binding the SSL/TLS layer to the application layer (i.e., channel binding [197, 10]). As a result, a MITM adversary trying to avoid detection by modifying the DCL is not only forced to compromise a CA to obtain a forged certificate but also to compromise each of the targeted domains to obtain users' authentication credentials.

After a DVCert transaction, the browser can verify the authenticity and integrity of the received DCL based on the user's password. A cryptographic hash of the DCL

is included in the PAK protocol computations; thus, any unauthorized changes to the DCL will be detected by the browser. In addition, the server is also authenticated once the DVCert transaction is completed because it is based on the users password. The cryptographic hash function used to compute the hash value of the DCL and the certificate's fingerprint needs to be collision resistant. Given the reported collisions attacks against MD5 [195] and SHA-1 [194], we recommend the use of a more robust hash algorithm such as SHA-256 or SHA-512.

An adversary can try to capture DVCert messages and use offline attacks to obtain user authentication credentials. However, the attacker needs to execute a MITM attack first to access DVCert messages. Thus, such attempts will be detected by DVCert. Furthermore, PAK's formal proofs of security for standard [26] and short exponents [127] (i.e., 384 bits) provide strong guarantees that the adversary will not learn password information from DVCert messages. DVCert modifications to PAK do not affect these proofs. For example, PAK and DVCert transmit the same number of hash values (2) over the network. The main difference is that DVCert uses one message less and uses the DCL as part of the computation of h_1.

We used ProVerif [22, 21], an automatic cryptographic protocol verifier, to formally characterize DVCert. *Using ProVerif, we successfully demonstrated that DVCert does not leak password information (i.e., resilience to offline attacks).* The DVCert model in pi calculus and the output results from ProVerif are shown in the Appendix B.

Because DVCert does not provide user authentication, the credentials stored in the browser or the server can be used to masquerade as the server but not as the user. Therefore, DVCert offers resilience to server compromise similar to augmented PAKE protocols. The adversary can still use offline dictionary attacks against the stolen credentials, but the use of strong passwords can mitigate this risk.

The DCL includes fingerprints of certificates from third-party domains because

these certificates cannot be validated directly (users do not share secrets with these domains). This is important because a MITM attack against a third-party SSL/TLS connection could be used to compromise the session with the web application (e.g., code injection attacks). The web application is responsible for maintaining the latest certificate information from third-party domains in the DCL. For example, the web application could rely on existing secure connections with third-party domains to obtain their certificate information. Alternatively, the application could rely on third-party validation mechanisms (e.g., network notaries).

A concern with PAKE protocols is the risk of denial of service attacks due to the cost of public key operations. DVCert mitigates this risk by optimizing such operations without reducing security. For example, DVCert can use shorter exponents for better performance without affecting formal proofs of security. PAK allows the use of exponents with a minimum size of 384 bits (1024 bits DH group) [127] while maintaining a similar level of security. Another suggested optimization is the use of static parameters in the server (i.e., b, g^b and m_2) to reduce the number of operations (see Section 7.4). This technique affects the protocol's perfect forward secrecy property; however, DVCert does not require it (i.e., the session secret is not used for encryption). Finally, the web application could also monitor and limit the number of DVCert requests a user can make per day according to a domain policy.

7.4 Experimental Analysis

We develped DVCert browser and server components (see Figure 38) to evaluate their performance and deployability. The DVCert browser component was implemented as an extension for Firefox 10.0.x and Firefox for mobile (Fennec) 4.03b. The

extensions were written mainly in Javascript, but we also used C code for modular exponentiation operations through Firefox's js-ctypes API and the GNU Multiple Precision Arithmetic library (GMP) [2]. Approximately 500 lines of code were required for both extensions. The DVCert server component was implemented in PHP and required approximately 400 lines of code. More importantly, the DVCert server component is completely independent of the web application code; only access to the user database is required. PAK implementation details as well as test vectors were obtained from the RFC 5683 [29] and the ITU-T Recommendation X.1035 [98]. The experiments used a laptop (Apple MacBook Pro with dual core 2.53 GHz processor, 4GB of memory and Mac OS X 10.6) and a smartphone (Samsung Galaxy S 4G with a 1 GHz Cortex-A8 processor, 512 MB of memory and Android 2.2.1) as our clients. On the server side, we used a Ubuntu 10.10 server with 2 quad-core 2.00 GHz processors, 16 GB of memory and Gigabit Ethernet. The server was configured with Apache 2.2, PHP 5.3 and a 2048 bits RSA certificate. Finally, our prototype DVCert implementation is currently available for evaluation at `http://www.cc.gatech.edu/~idacosta/dvcert/index.html`.

Certificate validation operations using the DCL are inexpensive. For example, for each SSL/TLS connection, the browser executes one hash operation and one search operation. Assuming an ordered DCL, binary search is used to determine if a certificate is in the DCL with time $O(\log n)$, where the DCL's size n is in the order of tens of certificates. In addition, the size of the DCL is small (e.g., a SHA-1 certificate fingerprint requires only 160 bits). Hence, the impact on network bandwitdh due to the DCL is negligible. Therefore, our experimental evaluation focused on the costs associated with DVCert transactions where more complex operations take place.

[2] Javascript-only DVCert add-ons for Firefox required an execution time at least one order of magnitude higher than add-ons using C native code for modular exponentiation, particularly in the smartphone. Ultimately, we envision DVCert to be implemented directly in the browser and using native code for its operations.

First, we measured the time required to generate a DVCert request (t_g) and the time required to verify the corresponding response (t_v) in the browser for different exponent sizes: 2048, 1024 and 384 bits. Morevoer, we used a DCL with one certificate fingerprint in all the experiments. Table 9 shows the results for 100 DVCert transactions per configuration using a laptop and a smartphone, including 95% confidence intervals. The results show that for 2048 bits exponents, an often recommended size for standard key exchange protocols [23], the browser required 26.78 ms and 440.58 ms of total computation time ($t_g + t_v$) on the laptop and on the smartphone respectively. While these computation times should not affect the user experience due to the low frequency of DVCert transactions, we can see that using 384 bits exponents decreased these times to 12.03 ms on the laptop (55.07% improvement) and 97.70 ms on the smartphone (77.82% improvement); thus, reducing the chance that users may notice these operations.

Second, we measured the server response time using network traces for single HTTPS requests (baseline) and HTTPS requests with DVCert. Each request retrieved a small HTML page ($\approx$ 500 bytes. We chose this small size to measured only the overhead added by SSL/TLS and DVCert). Moreover, our measurements did not include SSL/TLS setup times. For HTTPS request with DVCert, we evaluated different exponent sizes (2048, 1024 and 384 bits) and the use of dynamic (t_r) and static (t_{rsp}) server parameters. Based on these measurements, we estimated how much time the server spent on DVCert operations (t_d and t_{dsp}) by subtracting the baseline time from the HTTPS+DVCert server response times. The results for 100 DVCert transactions per configuration are shown in Table 10, including 95% confidence intervals. The most robust configuration, 2048 bits and dynamic parameters, required 10.71 ms of additional server computation time, while the most efficient configuration, 384 bits and static parameters, required around 0.54 ms (94.96% improvement). Thus, the most efficient DVCert configuration requires less time than serving a HTTPS request

Table 9: DVCert request generation time (t_g) and response verification time (t_v), including 95% confidence intervals, on a laptop and on a smartphone for different exponent sizes. For 384 bits exponents, a DVCert transaction required a total time ($t_g + t_v$) of 12.03 ms on the laptop and 97.70 ms on the smartphone. Thus, these operations are unlikely to be noticed by users

Exponent Size (bits)	**Laptop t_g (ms)**	**Laptop t_v (ms)**	**Smartphone t_g (ms)**	**Smartphone t_v (ms)**
2048	10.36 (±0.0941)	16.42 (±0.2883)	171.92 (±1.7883)	268.66 (±9.6384)
1024	3.95 (±0.0693)	9.55 (±0.1358)	48.68 (±2.1108)	71.88 (±7.8691)
384	3.26 (±0.0860)	8.77 (±0.1382)	33.58 (±0.7279)	64.12 (±7.4392)

Table 10: Server response time (t_r) for a single HTTPS request (baseline) and single HTTPS requests with DVCert using dynamic and static parameters (t_{rsp}) and different exponent sizes. By subtracting the time of a single HTTPS request, we estimated the cost of DVCert operations with static (t_d) and dynamic (t_{dsp}) parameters and determined the percentage of improvement (% Imp.) due to static parameters. For 384 bits and static parameters, DVCert operations required half of the time used to server a single HTTPS request.

Request Type	**t_r (ms)**	**t_d (ms)**	**t_{rsp} (ms)**	**t_{dsp} (ms)**	**% Imp. (t_{dsp})**
HTTPS	1.17 (±0.0140)	–	1.17 (±0.0140)	–	–
HTTPS + DVCert 2048 bits	11.88 (±0.0064)	10.71	6.66 (±0.0066)	5.49	48.74%
HTTPS + DVCert 1024 bits	3.02 (±0.0060)	1.85	2.20 (±0.0056)	1.03	44.32%
HTTPS + DVCert 384 bits	2.04 (±0.0084)	0.87	1.71 (±0.0060)	0.54	37.93%

(1.17 ms) and it is smaller than the average network jitter in the US (0.67 ms [11]). Also, Table 10 shows how static parameters can reduce DVCert processing time on the server by at least 38%. Overall, these results show that DVCert operations have similar processing requirements to other server operations (e.g., SSL/TLS setup, HTTPS requests processing) while still maintaining robust security guarantees. Thus, it is unlikely that DVCert could degrade server performance or increase the risk of DoS attacks.

Finally, we evaluated the overall impact of DVCert on server throughput in the hypothetical scenario where each SSL/TLS connection includes a DVCert transaction

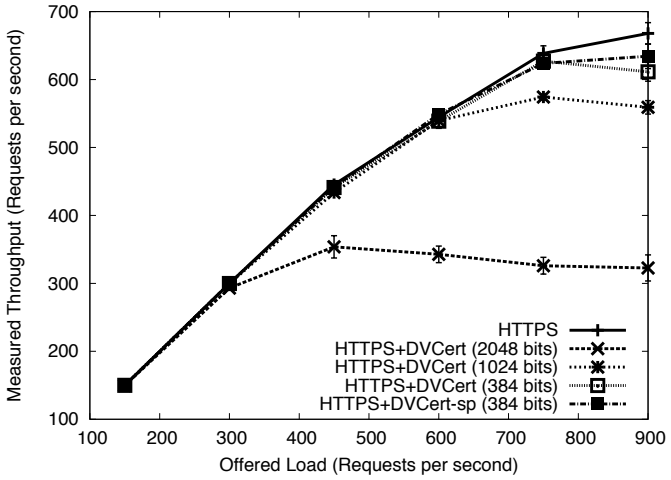


Figure 40: Comparison of the web server throughput for single HTTPS request and HTTPS requests with DVCert in the hypothetical case that DVCert transactions are executed per SSL/TLS connection (i.e., upper bound). HTTPS+DVCert configurations used different exponent sizes and one configuration used static parameters (HTTPS+DVCert-sp). Using DVCert with 384 bits exponents allowed a maximum throughput close to the one achieved with single HTTPS requests. Thus, DVCert transactions are unlikely to degrade server performance. Note: SSL/TLS connections used a 2048 bits RSA key.

(i.e., upper bound). For this purpose, we measured the rate of HTTPS requests (using one SSL/TLS connection per request) and the rate of HTTPS+DVCert requests that the server can handle. As before, we evaluated DVCert with different exponent sizes (2048, 1024 and 384 bits) and one setup with static parameters and 384 bits exponents. The test load was generated with *httperf*, a HTTP traffic generator tool. Figure 40 shows the results of this experiment for 10 measurements per point (300 in total), including 95% confident intervals. This figure shows that, even if every SSL/TLS connection uses a DVCert transaction, using 384 bits exponents allows a maximum throughput close to the one obtained using single HTTPS requests. Moreover, 1024 bit exponents could also allow a similar performance if static parameters are used (based on the results shown in Table 10). Thus, using 1024 bits exponents or shorter and static parameters reduces the risk of DoS attacks, eliminating the need

for additional DoS defenses (e.g., client puzzles).

7.5 Discussion

7.5.1 DVCert Benefits

In addition to meeting the design goals described in Section 7.2.2, DVCert solves most of the problems hindering the deployment of third-party defenses against MITM attacks (see Section 7.1.3). First, *DVCert is easier to deploy and maintain.* In most scenarios, DVCert should not require additional infrastructure due to its low processing costs. Only minor modifications are required to add DVCert support to the web application and the browser (see Figure 38). For example, DVCert only needs access to the application's user database and certificate information (i.e., the DCL). Hence, DVCert can be deployed as an independent service without modifying any existing functionality in the application. In the browser, DVCert can also be implemented as an independent component that only requires the certificate information used on each SSL/TLS connection and secure storage for the password salted hashes and DCL data. Moreover, by relying on passwords, users do not need to deal with additional secrets or devices and can benefit from DVCert on a wider range of platforms. Second, *DVCert has a simpler trust model.* It relies on existing trust relationships between users and web applications; hence, users do not need to assess and establish new trust relationships with third-parties. Third, *DVCert does not introduce new privacy risks.* User browsing activity is not revealed to third-parties when a certificate is validated using DVCert. This property is particularly important for users with high privacy and anonymity requirements (e.g., Tor users). Fourth, *certificate revocation is simpler.* For instance, a certificate can be revoked by just removing it from the DCL. Thus, there is no need for mechanisms such as CRLs and OCSP, both criticised due to their ineffectiveness [117]. Fifth, *DVCert is more resilient to compromise than third-party approaches.* Third-party solutions can vouch for certificates belonging to

a large number of domains. However, if compromised, then all the protected domains could be affected by MITM attacks. In contrast, DVCert is deployed independently per domain; thus, attacks against one domain will not affect other domains. Finally, *DVCert is compatible with captive portals in certain scenarios.* For instance, DVCert could verify the certificates of captive portals that already share a secret with the user (e.g., account with a Wi-Fi provider) or where the user receives a shared secret via a secondary channel (e.g., a paper receipt).

7.5.2 DVCert Limitations

DVCert allows web applications to vouch for their certificates using existing authentication credentials. Thus, DVCert can only protect web applications where the user has an account and a shared secret. However, this is not a major limitation because most of the web applications that are likely to be targeted by adversaries (e.g., sites with private information) require authentication credentials. A related case are web applications that rely on federated identity management (e.g., OpenID) or Single sign-on (SSO) systems. Here, users share a password with an identity provider instead of the web application. Still, DVCert can be extended to validate certificates in such scenarios. For instance, the web application can provide its DCL to the identity provider during the login process. Then, the browser can execute a DVCert transaction to obtain not only the DCL of the identity provider but also of the targeted application. We plan to explore this idea in our future work. Another limitation is that DVCert cannot be used to protect the first connection to a web application. DVCert is by design a trust-on-first-use (TOFU) [196] mechanism such as the SSH protocol. Therefore, when registering to a web application for the first time, users can only rely on CA signatures and other third-party mechanisms to validate certificates. However, for most scenarios, it is unlikely that adversaries will be monitoring users before they have created an account with a web application. Moreover, applications

with high security requirements could also use secondary channels to protect the user registration process.

7.6 Summary

As recent incidents have demonstrated, adversaries are exploiting weaknesses in the CA trust model to compromise communications protected by SSL/TLS via MITM attacks. This trend is likely to accelerate as more and more web applications adopt SSL/TLS to protect all their communications. Currently proposed solutions face multiple challenges due to their complexity and deployment and operational costs; thus, they are unlikely to be widely available in the near future. We present DVCert, a practical mechanism that relies on previously established shared secrets to allow the web application to directly and securely vouch for the authenticity of its certificates. By using a single round-trip transaction with the web application, based on a modified PAK protocol, the browser learns the information required to locally verify all the certificates that could be used during a session with the application. Our experimental analysis shows that DVCert transactions require little execution time on the server and the browser; therefore, they should not have a serious impact on server performance or user experience. Finally, DVCert could be extended to protect not only the integrity of SSL/TLS certificates but also other application's resources such as Javascript code and binary objects. We intend to explore this approach in our future work.

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

Internet systems such as VoIP and Web applications will continue growing in size and complexity to support a larger number of users and richer functionality. Mobile platforms such as smartphones are rapidly becoming the main medium to access and consume Internet content. This trend means that users will be generating more requests to Internet applications due to the always-connected nature of smartphones. Similarly, the adoption of ubiquitous computing technologies (e.g., smart devices, wearable computing, in-car computing, etc.) will also increase the number and type of requests that need to be processed by Internet applications. All the request load generated by applications needs to be properly handled by taking into account not only the requirements of Internet applications but also the constraints of clients' platforms. As the threat level against Internet application increases and powerful adversaries try to compromise these systems, the security of Internet applications cannot longer be considered a secondary goal. Therefore, more robust security mechanisms that satisfy the performance and scalability of large-scale Internet applications are needed.

In this dissertation we have demonstrated that there is no inherent conflict between implementing robust authentication protocols and the unprecedented performance and scalability requirements of large-scale Internet applications. We have shown that by taking into account factors such as network latency, server state requirements, network bandwidth, response times and deployment costs, we can design and implement practical authentication protocols that offer stronger security guarantees than currently deployed mechanisms, while satisfying the performance and

scalability constraints of large-scale VoIP and Web applications.

The main contributions of this dissertation are:

- **Experimental analysis of the impact of SIP Digest authentication on a large-scale VoIP infrastructure and design and evaluation of more robust and efficient alternative.** We demonstrated that the seemingly efficient SIP Digest authentication protocol, the most popular authentication protocol for VoIP, reduced proxy's performance by almost three orders of magnitude in our distributed SIP testbed. Therefore, we designed Proxychain, a mechanism that provides strong authentication between VoIP providers and their customers. Unlike previously deployed mechanisms, Proxychain is highly scalable and offers throughput improvements of greater than an order of magnitude. This increased efficiency allows providers not only to support a much larger customer base on a relatively limited hardware footprint, but also increases the overall security of the network by allowing for multiple message types to be authenticated

- **Design, implementation and evaluation of a robust and practical HTTP session authentication protocol resistant to session hijacking attacks**. As an alternative to the inherently insecure use of HTTP cookies as session authentication tokens and to mitigate the threat of session hijacking attacks, we developed One-Time Cookies (OTC), a more secure alternative to authentication cookies that does not require state in the web application. OTC is not only resistant to session hijacking, but also maintains the simplicity and performance benefits of cookies. More critically, OTC addresses the shortcomings of previously proposed solutions by removing the need for state in the web application. In addition, OTC offers another security layer to web applications that already support always-on HTTPS by reducing the threats associated with

cookies.

- **Design, implementation and evaluation of an efficient and easy to deploy protocol that provides stronger server authentication in SSL/TLS connections and prevents MITM attacks.** We presented DVCert, a practical mechanism that relies on previously established shared secrets to allow the web application to directly and securely vouch for the authenticity of its certificates (i.e., server authentication). By using a single round-trip transaction with the web application, based on a modified PAK protocol, the browser learns the information required to locally verify all the certificates that could be used during a session with the application. As a result, to execute a MITM attack, an adversary not only needs to compromise a CA but also each targeted web domain.

8.1 Future Work

Our work can be extended in a variety of ways:

- *Security and efficiency improvements to Proxychain.* While Proxychain provides better security guarantees than SIP Digest authentication, it is still vulnerable to offline dictionary attacks. By using ideas from our work with DVCert, we can make Proxychain resistant to offline attacks by adding low-cost PAKE operations. In addition, we could also explore the use of encrypted tickets (as in OTC) to replace hash chains and avoid additional state in the SIP proxy and hash chain initialization operations.

- *Improving and extending OTC.* OTC offers no defense against CSRF attacks. For example, if an adversary manages to inject malicious code in the browser (e.g., XSS attack), she could be able to generate certain requests (OTC only prevents the adversary to steal the session secrets in this scenario). To prevent

this type of attacks, OTC can be extended with anti-CSRF techniques. In particular, OTC could be combined with an extended version of DVCert (see next point) to prevent the execution of injected code. Moreover, OTC could also be extended to not only protect the integrity of the user requests but also the integrity of the server responses.

- *Extending DVCert protection.* DVCert can be extended to protect not only the integrity of the web application's certificates but also the integrity of other resources such as JavaScript code and HTML login forms. This approach can help to prevent phishing and XSS attacks.

- *DVCert support for Federated authentication.* DVCert can be extended to work in scenarios where federated authentication (e.g., OpenID) and Single-Sign On (SSO) mechanisms are used. Thus, DVCert will be able to provide protection to a larger number of users and Web applications.

APPENDIX A

OTC SECURITY VERIFICATION USING PROVERIF V.1.86

A.1 Pi Calculus modeling of OTC

```
type key.
type nonce.
type timestamp.
free HTTPS:channel [private].
free HTTP:channel.
free secret1_ks, secret2_ks, secret_kw:bitstring [private].
fun encrypt(bitstring,key): bitstring.
reduc forall m: bitstring, k: key; decrypt(encrypt(m,k),k) = m.
fun hmac(bitstring, key): bitstring.
fun xor(key, nonce):key.
(* Secrecy queries *)
query attacker(secret1_ks).
query attacker(secret2_ks).
query attacker(secret_kw).
(* Correspondance events *)
event acceptsClient(key).
event termServer(key).
(* Browser to WebApp session authentication query *)
query x:key; inj-event(termServer(x)) ==>inj-event(acceptsClient(x)).
(* Browser macro *)
let browser(uid:bitstring, passwd:bitstring) =
    out(HTTPS, (uid, passwd));
    in(HTTPS, (cid:bitstring, ns:nonce, ks:key, ts:timestamp, ticket:bitstring));
    event acceptsClient(ks);
    new url:bitstring;
    new th:timestamp;
    new data:bitstring;
    out(HTTP, (ns, th, hmac((url, th, data), ks), ticket, url, data)).
(* Web Application macro *)
let webapp(kw:key)=
```

```
    in(HTTPS, (uid:bitstring, passwd:bitstring));
    new ks:key;
    new ns:nonce;
    new ts:timestamp;
    new cid:bitstring;
    out(HTTPS, (cid, ns, ks, ts, encrypt((cid, uid, ks, ts), xor(kw,ns))));
    in(HTTP, (nsx:nonce, th:timestamp, hmacx:bitstring, ticket:bitstring,
       url:bitstring, data:bitstring));
    let (=cid, =uid, ksx:key, tsx:bitstring) = decrypt(ticket, xor(kw, nsx)) in
    out(HTTP, encrypt(secret_kw, kw));
    out(HTTP, encrypt(secret1_ks, ks));
    out(HTTP, encrypt(secret2_ks, ksx));
    let (hmacy:bitstring) = hmac((url,th,data), ksx) in
    if hmacx = hmacy then event termServer(ks).
(* Main *)
process
    new kw:key;
    new uid:bitstring;
    new passwd:bitstring;
    ( !(browser(uid, passwd)) |  !(webapp(kw)) )
```

A.2 Proverif Output

```
RESULT inj-event(termServer(x)) ==> inj-event(acceptsClient(x)) is true.
-- Query not attacker(secret_kw[])
Completing...
Starting query not attacker(secret_kw[])
RESULT not attacker(secret_kw[]) is true.
-- Query not attacker(secret2_ks[])
Completing...
Starting query not attacker(secret2_ks[])
RESULT not attacker(secret2_ks[]) is true.
-- Query not attacker(secret1_ks[])
Completing...
Starting query not attacker(secret1_ks[])
RESULT not attacker(secret1_ks[]) is true.
```

APPENDIX B

DVCERT SECURITY VERIFICATION USING PROVERIF V.1.86

B.1 Pi Calculus modeling of DVCert

```
(* DVCert protocol based on PAK description in RFC 5683 *)

free c: channel.
type exponent.
fun G_to_bitstring(G):bitstring [data, typeConverter].
fun bitstring_to_G(bitstring):G [data, typeConverter].

(* Diffie-Hellman *)
const g: G.
fun exp(G, exponent): G.
equation forall x: exponent, y: exponent; exp(exp(g, x), y) = exp(exp(g, y), x).

(* Modular multiplication and division *)
fun multm(G, G):G.
fun divm(G, G):G.
equation forall x:G, y:G; divm(multm(x,y),y) = x.
equation forall x:G, y:G; multm(divm(x,y),y) = x.

(* Hash function *)
fun h(bitstring):bitstring.

(* Host names *)
const A, B: bitstring.
const s1, s2, s3, s4: bitstring.

free PAB, PAA, PBB: bitstring [private].
weaksecret PAB.
weaksecret PAA.
weaksecret PBB.
```

```
(* Initiator with identity hostA talking to responder with identity hostX *)
(* Browser *)
let processA(hostA: bitstring, hostX: bitstring, P: bitstring) =
    new RA: exponent;
    let gRA = exp(g, RA) in
    let m1 = G_to_bitstring(multm(gRA, bitstring_to_G(h((s1, hostA, hostX, P))))) in
    out(c, (hostA, m1));
    in(c, (m2:bitstring, h1:bitstring, h2:bitstring, DCL:bitstring));
    let gRB = divm(bitstring_to_G(m2), bitstring_to_G(h((s2, hostA, hostX, P)))) in
    let K = G_to_bitstring(exp(gRB, RA)) in
    if h1 = h((s3, hostA, hostX, P, G_to_bitstring(gRA), G_to_bitstring(gRB), K, h(DCL)))
&& h2 = h((s3, hostA, hostX, P, G_to_bitstring(gRA), G_to_bitstring(gRB), K)) then
    0.

(* Server *)
let processB(hostB: bitstring, hostX: bitstring, P: bitstring, DCL: bitstring) =
    in(c, (=hostX, m1: bitstring));
    new RB: exponent;
    let gRA = divm(bitstring_to_G(m1), bitstring_to_G(h((s1, hostX, hostB, P)))) in
    let gRB = exp(g, RB) in
    let K = G_to_bitstring(exp(gRA, RB)) in
    let m2 = G_to_bitstring(multm(gRB, bitstring_to_G(h((s2, hostX, hostB, P))))) in
    let h1 = h((s3, hostX, hostB, P, G_to_bitstring(gRA), G_to_bitstring(gRB), K, h(DCL))) in
    let h2 = h((s3, hostX, hostB, P, G_to_bitstring(gRA), G_to_bitstring(gRB), K)) in
    out(c, (m2, h1, h2)).

process
    new DCL:bitstring;
    (!processA(A, A, PAA)) |
    (!processB(A, A, PAA, DCL)) |
    (!processA(B, B, PBB)) |
    (!processB(B, B, PBB, DCL)) |
    (!processA(A, B, PAB)) |
    (!processB(A, B, PAB, DCL)) |
    (!processA(B, A, PAB)) |
    (!processB(B, A, PAB, DCL))
```

B.2 Proverif Output

```
Linear part:
```

```
exp(exp(g,x),y) = exp(exp(g,y),x)
Completing equations...
Completed equations:
exp(exp(g,x),y) = exp(exp(g,y),x)
Convergent part:
multm(divm(x_6,y_7),y_7) = x_6
divm(multm(x_4,y_5),y_5) = x_4
Completing equations...
Completed equations:
divm(multm(x_4,y_5),y_5) = x_4
multm(divm(x_6,y_7),y_7) = x_6
Completed destructors:
not(false) => true
not(true) => false
Process:
{1}new DCL: bitstring;
(
    {2}!
    {3}new RA: exponent;
    {4}let gRA: G = exp(g,RA) in
    {5}let m1: bitstring = multm(gRA,h((s1,A,A,PAA))) in
    {6}out(c, (A,m1));
    {7}in(c, (m2: bitstring,h1: bitstring,h2: bitstring,DCL_8: bitstring));
    {8}let gRB: G = divm(m2,h((s2,A,A,PAA))) in
    {9}let K: bitstring = exp(gRB,RA) in
    {10}if h1 = h((s3,A,A,PAA,gRA,gRB,K,h(DCL_8))) then
    {11}if h2 = h((s3,A,A,PAA,gRA,gRB,K)) then
    0
) | (
    {12}!
    {13}in(c, (=A,m1_9: bitstring));
    {14}new RB: exponent;
    {15}let gRA_10: G = divm(m1_9,h((s1,A,A,PAA))) in
    {16}let gRB_11: G = exp(g,RB) in
    {17}let K_12: bitstring = exp(gRA_10,RB) in
    {18}let m2_13: bitstring = multm(gRB_11,h((s2,A,A,PAA))) in
    {19}let h1_14: bitstring = h((s3,A,A,PAA,gRA_10,gRB_11,K_12,h(DCL))) in
    {20}let h2_15: bitstring = h((s3,A,A,PAA,gRA_10,gRB_11,K_12)) in
    {21}out(c, (m2_13,h1_14,h2_15))
) | (
    {22}!
```

```
    {23}new RA_16: exponent;
    {24}let gRA_17: G = exp(g,RA_16) in
    {25}let m1_18: bitstring = multm(gRA_17,h((s1,B,B,PBB))) in
    {26}out(c, (B,m1_18));
    {27}in(c, (m2_19: bitstring,h1_20: bitstring,h2_21: bitstring,DCL_22: bitstring));
    {28}let gRB_23: G = divm(m2_19,h((s2,B,B,PBB))) in
    {29}let K_24: bitstring = exp(gRB_23,RA_16) in
    {30}if h1_20 = h((s3,B,B,PBB,gRA_17,gRB_23,K_24,h(DCL_22))) then
    {31}if h2_21 = h((s3,B,B,PBB,gRA_17,gRB_23,K_24)) then
    0
) | (
    {32}!
    {33}in(c, (=B,m1_25: bitstring));
    {34}new RB_26: exponent;
    {35}let gRA_27: G = divm(m1_25,h((s1,B,B,PBB))) in
    {36}let gRB_28: G = exp(g,RB_26) in
    {37}let K_29: bitstring = exp(gRA_27,RB_26) in
    {38}let m2_30: bitstring = multm(gRB_28,h((s2,B,B,PBB))) in
    {39}let h1_31: bitstring = h((s3,B,B,PBB,gRA_27,gRB_28,K_29,h(DCL))) in
    {40}let h2_32: bitstring = h((s3,B,B,PBB,gRA_27,gRB_28,K_29)) in
    {41}out(c, (m2_30,h1_31,h2_32))
) | (
    {42}!
    {43}new RA_33: exponent;
    {44}let gRA_34: G = exp(g,RA_33) in
    {45}let m1_35: bitstring = multm(gRA_34,h((s1,A,B,PAB))) in
    {46}out(c, (A,m1_35));
    {47}in(c, (m2_36: bitstring,h1_37: bitstring,h2_38: bitstring,DCL_39: bitstring));
    {48}let gRB_40: G = divm(m2_36,h((s2,A,B,PAB))) in
    {49}let K_41: bitstring = exp(gRB_40,RA_33) in
    {50}if h1_37 = h((s3,A,B,PAB,gRA_34,gRB_40,K_41,h(DCL_39))) then
    {51}if h2_38 = h((s3,A,B,PAB,gRA_34,gRB_40,K_41)) then
    0
) | (
    {52}!
    {53}in(c, (=B,m1_42: bitstring));
    {54}new RB_43: exponent;
    {55}let gRA_44: G = divm(m1_42,h((s1,B,A,PAB))) in
    {56}let gRB_45: G = exp(g,RB_43) in
    {57}let K_46: bitstring = exp(gRA_44,RB_43) in
    {58}let m2_47: bitstring = multm(gRB_45,h((s2,B,A,PAB))) in
```

```
    {59}let h1_48: bitstring = h((s3,B,A,PAB,gRA_44,gRB_45,K_46,h(DCL))) in
    {60}let h2_49: bitstring = h((s3,B,A,PAB,gRA_44,gRB_45,K_46)) in
    {61}out(c, (m2_47,h1_48,h2_49))
) | (
    {62}!
    {63}new RA_50: exponent;
    {64}let gRA_51: G = exp(g,RA_50) in
    {65}let m1_52: bitstring = multm(gRA_51,h((s1,B,A,PAB))) in
    {66}out(c, (B,m1_52));
    {67}in(c, (m2_53: bitstring,h1_54: bitstring,h2_55: bitstring,DCL_56: bitstring));
    {68}let gRB_57: G = divm(m2_53,h((s2,B,A,PAB))) in
    {69}let K_58: bitstring = exp(gRB_57,RA_50) in
    {70}if h1_54 = h((s3,B,A,PAB,gRA_51,gRB_57,K_58,h(DCL_56))) then
    {71}if h2_55 = h((s3,B,A,PAB,gRA_51,gRB_57,K_58)) then
    0
) | (
    {72}!
    {73}in(c, (=A,m1_59: bitstring));
    {74}new RB_60: exponent;
    {75}let gRA_61: G = divm(m1_59,h((s1,A,B,PAB))) in
    {76}let gRB_62: G = exp(g,RB_60) in
    {77}let K_63: bitstring = exp(gRA_61,RB_60) in
    {78}let m2_64: bitstring = multm(gRB_62,h((s2,A,B,PAB))) in
    {79}let h1_65: bitstring = h((s3,A,B,PAB,gRA_61,gRB_62,K_63,h(DCL))) in
    {80}let h2_66: bitstring = h((s3,A,B,PAB,gRA_61,gRB_62,K_63)) in
    {81}out(c, (m2_64,h1_65,h2_66))
)

-- Weak secret PBB
Termination warning: v_716 <> v_717 && attacker_guess(v_715,v_716)
    && attacker_guess(v_715,v_717) -> bad
Selecting 0
Termination warning: v_719 <> v_720 && attacker_guess(v_719,v_718)
    && attacker_guess(v_720,v_718) -> bad
Selecting 0
Completing...
Termination warning: v_716 <> v_717 && attacker_guess(v_715,v_716)
    && attacker_guess(v_715,v_717) -> bad
Selecting 0
Termination warning: v_719 <> v_720 && attacker_guess(v_719,v_718)
    && attacker_guess(v_720,v_718) -> bad
```

```
Selecting 0
Termination warning: v_3491 <> v_3492 && attacker(v_3491)
    && attacker_guess(v_3491,v_3492) -> bad
Selecting 1
Termination warning: v_3695 <> v_3696 && attacker(v_3695)
    && attacker_guess(v_3696,v_3695) -> bad
Selecting 1
200 rules inserted. The rule base contains 186 rules. 87 rules in the queue.
400 rules inserted. The rule base contains 346 rules. 150 rules in the queue.
600 rules inserted. The rule base contains 500 rules. 198 rules in the queue.
800 rules inserted. The rule base contains 682 rules. 296 rules in the queue.
1000 rules inserted. The rule base contains 880 rules. 346 rules in the queue.
1200 rules inserted. The rule base contains 1068 rules. 331 rules in the queue.
1400 rules inserted. The rule base contains 1260 rules. 230 rules in the queue.
1600 rules inserted. The rule base contains 1368 rules. 142 rules in the queue.
1800 rules inserted. The rule base contains 1518 rules. 102 rules in the queue.
2000 rules inserted. The rule base contains 1709 rules. 133 rules in the queue.
2200 rules inserted. The rule base contains 1908 rules. 126 rules in the queue.
2400 rules inserted. The rule base contains 2107 rules. 112 rules in the queue.
2600 rules inserted. The rule base contains 2280 rules. 56 rules in the queue.
RESULT Weak secret PBB is true (bad not derivable).
-- Weak secret PAA
Termination warning: v_2761076 <> v_2761077 && attacker_guess(v_2761075,v_2761076)
    && attacker_guess(v_2761075,v_2761077) -> bad
Selecting 0
Termination warning: v_2761079 <> v_2761080 && attacker_guess(v_2761079,v_2761078)
    && attacker_guess(v_2761080,v_2761078) -> bad
Selecting 0
Completing...
Termination warning: v_2761076 <> v_2761077 && attacker_guess(v_2761075,v_2761076)
    && attacker_guess(v_2761075,v_2761077) -> bad
Selecting 0
Termination warning: v_2761079 <> v_2761080 && attacker_guess(v_2761079,v_2761078)
    && attacker_guess(v_2761080,v_2761078) -> bad
Selecting 0
Termination warning: v_2763813 <> v_2763814 && attacker(v_2763813)
    && attacker_guess(v_2763813,v_2763814) -> bad
Selecting 1
Termination warning: v_2764017 <> v_2764018 && attacker(v_2764017)
    && attacker_guess(v_2764018,v_2764017) -> bad
Selecting 1
```

```
200 rules inserted. The rule base contains 186 rules. 87 rules in the queue.
400 rules inserted. The rule base contains 346 rules. 150 rules in the queue.
600 rules inserted. The rule base contains 500 rules. 198 rules in the queue.
800 rules inserted. The rule base contains 682 rules. 296 rules in the queue.
1000 rules inserted. The rule base contains 880 rules. 346 rules in the queue.
1200 rules inserted. The rule base contains 1068 rules. 331 rules in the queue.
1400 rules inserted. The rule base contains 1260 rules. 230 rules in the queue.
1600 rules inserted. The rule base contains 1326 rules. 142 rules in the queue.
1800 rules inserted. The rule base contains 1518 rules. 102 rules in the queue.
2000 rules inserted. The rule base contains 1709 rules. 133 rules in the queue.
2200 rules inserted. The rule base contains 1908 rules. 126 rules in the queue.
2400 rules inserted. The rule base contains 2107 rules. 112 rules in the queue.
2600 rules inserted. The rule base contains 2280 rules. 56 rules in the queue.
RESULT Weak secret PAA is true (bad not derivable).
-- Weak secret PAB
Termination warning: v_5521398 <> v_5521399 && attacker_guess(v_5521397,v_5521398)
    && attacker_guess(v_5521397,v_5521399) -> bad
Selecting 0
Termination warning: v_5521401 <> v_5521402 && attacker_guess(v_5521401,v_5521400)
    && attacker_guess(v_5521402,v_5521400) -> bad
Selecting 0
Completing...
Termination warning: v_5521398 <> v_5521399 && attacker_guess(v_5521397,v_5521398)
    && attacker_guess(v_5521397,v_5521399) -> bad
Selecting 0
Termination warning: v_5521401 <> v_5521402 && attacker_guess(v_5521401,v_5521400)
    && attacker_guess(v_5521402,v_5521400) -> bad
Selecting 0
Termination warning: v_5524135 <> v_5524136 && attacker(v_5524135)
    && attacker_guess(v_5524135,v_5524136) -> bad
Selecting 1
Termination warning: v_5524339 <> v_5524340 && attacker(v_5524339)
    && attacker_guess(v_5524340,v_5524339) -> bad
Selecting 1
200 rules inserted. The rule base contains 186 rules. 87 rules in the queue.
400 rules inserted. The rule base contains 346 rules. 150 rules in the queue.
600 rules inserted. The rule base contains 500 rules. 194 rules in the queue.
800 rules inserted. The rule base contains 682 rules. 295 rules in the queue.
1000 rules inserted. The rule base contains 880 rules. 342 rules in the queue.
1200 rules inserted. The rule base contains 1068 rules. 330 rules in the queue.
1400 rules inserted. The rule base contains 1260 rules. 221 rules in the queue.
```

```
1600 rules inserted. The rule base contains 1292 rules. 117 rules in the queue.
1800 rules inserted. The rule base contains 1390 rules. 96 rules in the queue.
2000 rules inserted. The rule base contains 1574 rules. 105 rules in the queue.
2200 rules inserted. The rule base contains 1768 rules. 91 rules in the queue.
2400 rules inserted. The rule base contains 1951 rules. 43 rules in the queue.
RESULT Weak secret PAB is true (bad not derivable).
```

REFERENCES

[1] "Certificate Patrol," 2010.

[2] 3GPP, "ETSI TS 133 102 v7.1.0 Security Architecture," 2006.

[3] Abdelnur, H., Avanesov, T., Rusinowitch, M., and State, R., "Abusing SIP Authentication," in *Proceedings of the 2008 The Fourth International Conference on Information Assurance and Security*, pp. 237–242, IEEE Computer Society, 2008.

[4] Adams, C. and Farrell, S., "RFC 2510 - Internet X.509 Public Key Infrastructure Certificate Management Protocols," 1999.

[5] Adams, C. and Just, M., "PKI: Ten Years Later," in *PKI R&D Workshop*, 2004.

[6] Adida, B., "Beamauth: Two-Factor Web Authentication with a Bookmark," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[7] Adida, B., "Sessionlock: Securing Web Sessions Against Eavesdropping," in *Proceeding of the ACM International Conference on World Wide Web (WWW)*, 2008.

[8] Adya, A., Bolosky, W., Castro, M., Chaiken, R., Cermak, G., Douceur, J., Howell, J., Lorch, J., Theimer, M., and Wattenhofer, R., "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," in *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2002.

[9] Alicherry, M. and Keromytis, A. D., "DoubleCheck: Multi-path Verification Against Man-in-the-Middle Attacks," in *Proceedings of the IEEE Symposium on Computers and Communications*, 2009.

[10] Altman, J., Williams, N., and Zhu, L., "RFC 5929 - Channel Bindings for TLS," 2010.

[11] AT&T, "Network Averages," 2012.

[12] Aziz, B. and Hamilton, G., "Detecting Man-in-the-Middle Attacks by Precise Timing," in *Proceedings of the International Conference on Emerging Security Information, Systems and Technologies*, 2009.

[13] Balasubramaniyan, V. A., Acharya, A., Ahamad, M., Srivatsa, M., Dacosta, I., and Wright, C. P., "SERvartuka: Dynamic Distribution of State to Improve SIP Server Scalability," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pp. 562–572, 2008.

[14] Balasubramaniyan, V. A., Poonawalla, A., Ahamad, M., Hunter, M. T., and Traynor, P., "PinDr0p: using single-ended audio features to determine call provenance," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2010.

[15] Barth, A., "RFC 6265 - HTTP State Management Mechanism." https://tools.ietf.org/html/rfc6265, 2011.

[16] Bellare., M., "New proofs for NMAC and HMAC: Security without collision-resistance," in *Proceedings of Crypto*, 2006.

[17] Bellare, M. and Rogaway, P., "An introduction to modern cryptography (UCSD CSE 207 Course Notes)." `http://www.cse.ucsd.edu/$\sim$mihir/cse207/index.html`., 2005.

[18] Bellare, M., Canetti, R., and Krawczyk, H., "Keying Hash Functions for Message Authentication," in *Advances in Cryptology CRYPTO 96* (Koblitz, N., ed.), vol. 1109 of *Lecture Notes in Computer Science*, pp. 1–15, Springer Berlin / Heidelberg, 1996.

[19] Biddle, R., van Oorschot, P. C., Patrick, A. S., Sobey, J., and Whalen, T., "Browser interfaces and extended validation SSL certificates: an empirical study," in *Proceedings of the ACM workshop on Cloud computing security*, 2009.

[20] Black, J., Cochran, M., and Highland, T., "A Study of the MD5 Attacks: Insights and Improvements," in *Proceedings of the International Conference on Fast Software Encryption (FSE)*, 2006.

[21] Blanchet, B., "ProVerif: Cryptographic Protocol Verifier in the Formal Model." `http://www.proverif.ens.fr/`.

[22] Blanchet, B., "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *Proceedings of the IEEE Workshop on Computer Security Foundations (CSFW)*, 2001.

[23] BlueKrypt, "Cryptographic Key Length Recommendation," 2012.

[24] Blundo, C., Cimato, S., and Prisco, R. D., "A Lightweight Approach to Authenticated Web Caching," in *Proceedings of the Symposium on Applications and the Internet*, 2005.

[25] Bortz, A., Barth, A., and Czeskis, A., "Origin Cookies: Session Integrity for Web Applications," in *Proceedings of the Web 2.0 Security and Privacy Workshop (W2SP)*, 2011.

[26] Boyko, V., MacKenzie, P., and Patel, S., "Provably Secure Password-Authenticated Key Exchange using Diffie-Hellman," in *Proceedings of the International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT)*, 2000.

[27] Bright, P., "Microsoft Patches Major Hotmail 0-day Flaw After Apparently Widespread Exploitation." `http://arstechnica.com/information-technology/2012/04/microsoft-patches-major-hotmail-0-day-flaw-after-apparently-widespread-exploitation/`, 2012.

[28] Brown, M. A., "Traffic Control HOWTO Version 1.0.2." `http://linux-ip.net/articles/Traffic-Control-HOWTO/`, 2006.

[29] Brusilovsky, A., Faynberg, I., Zeltsan, Z., and Patel, S., "RFC 5683 - Password-Authenticated Key (PAK) Diffie-Hellman Exchange," 2010.

[30] BuddyPress, "BuddyPress.org." `http://buddypress.org/`.

[31] Butler, E., "Firesheep." `http://codebutler.com/firesheep`.

[32] CA/Browser Forum, "Guidelines For The Issuance And Management Of Extended Validation Certificates Version 1.3." `http://www.cabforum.org/Guidelines_v1_3.pdf`, 2010.

[33] Cao, F. and Jennings, C., "Providing Response Identity and Authentication in IP Telephony," in *Proceedings of the First International Conference on Availability, Reliability and Security (ARES)*, IEEE Computer Society, 2006.

[34] Cha, E.-C., Choi, H.-K., and Cho, S.-J., "Evaluation of Security Protocols for the Session Initiation Protocol," in *Proceedings of 16th International Conference on Computer Communications and Networks (ICCCN)*, pp. 611–616, IEEE, 2007.

[35] Chan, M., "China and Google: A Detailed Look." `http://blogs.aljazeera.net/asia/2011/03/23/china-and-google-detailed-look`, 2011.

[36] Charette, R., "DigiNotar Certificate Authority Breach Crashes e-Government in the Netherlands." `http://spectrum.ieee.org/riskfactor/telecom/security/diginotar-certificate-authority-breach-crashes-egovernment-in-the-netherlands/?utm_source=techalert&utm_medium=email&utm_campaign=091511`, 2011.

[37] Chen, S., Mao, Z., Wang, Y.-M., and Zhang, M., "Pretty-Bad-Proxy: An Overlooked Adversary in Browsers' HTTPS Deployments," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2009.

[38] Chen, T.-h., Yeh, H.-l., Liu, P.-c., Hsiang, H.-c., and Shih, W.-k., "A Secured Authentication Protocol for SIP Using Elliptic Curves Cryptography," in *Communication and Networking* (Kim, T.-h., Chang, A. C.-C., Li, M., Rong, C., Patrikakis, C. Z., and Ślzak, D., eds.), vol. 119 of *Communications in Computer and Information Science*, pp. 46–55, Springer Berlin Heidelberg, 2010.

[39] Choi, T. and Gouda, M. G., "HTTPI: An HTTP with Integrity," in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 2011.

[40] Close, T., "Waterken Server: Capability-based Security for the Web." `http://waterken.sourceforge.net/`, 1999.

[41] Coarfa, C., Druschel, P., and Wallach, D. S., "Performance Analysis of TLS Web Servers," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 1, 2006.

[42] ComScore, "Smartphones and Tablets Drive Nearly 7 Percent of Total U.S. Digital Traffic." `http://www.comscore.com/Press_Events/Press_Releases/2011/10/Smartphones_and_Tablets_Drive_Nearly_7_Percent_of_Total_U.S._Digital_Traffic`, 2011.

[43] Constantin, L., "XSS Attack on Twitter Subdomain Allowed for Complete Session Hijacking." `http://news.softpedia.com/news/XSS-Attack-on-Twitter-Subdomain-Allowed-Full-Session-Hijacking-148240.shtml`, 2010.

[44] Cortes, M., Ensor, J. R., and Esteban, J. O., "On SIP Performance," *Bell Labs Technical Journal*, vol. 9, pp. 155–172, 2004.

[45] Cortes, M., Esteban, J. O., and Jun, H., "Towards Stateless Core: Improving SIP Proxy Scalability," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, 2006.

[46] Cross, T., "Stealing Cookies with SSL Renegotiation." `http://blogs.iss.net/archive/stealingcookieswiths.html`, 2009.

[47] Dacosta, I., Balasubramaniyan, V., Ahamad, M., and Traynor, P., "Improving Authentication Performance of Distributed SIP Proxies," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 11, pp. 1804–1812, 2011.

[48] Dacosta, I., Ahamad, M., and Traynor, P., "Trust No One Else: Detecting MITM Attacks Against SSL/TLS Without Third-Parties," in *European Symposium on Research in Computer Security (ESORICS)*, 2012.

[49] Dacosta, I., Balasubramaniyan, V., Ahamad, M., and Traynor, P., "Improving Authentication Performance of Distributed SIP Proxies," in *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm)*, 2009.

[50] Dacosta, I., Chakradeo, S., Ahamad, M., and Traynor, P., "One-Time Cookies: Preventing Session Hijacking Attacks with Stateless Authentication Tokens," *ACM Transactions on Internet Technology (TOIT)*, 2012.

[51] Dacosta, I. and Traynor, P., "Proxychain: Developing a Robust and Efficient Authentication Infrastructure for Carrier-Scale VoIP Networks," in *Proceedings of the USENIX Annual Technical Conference*, 2010.

[52] Dhamija, R., Tygar, J. D., and Hearst, M., "Why Phishing Works," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006.

[53] Dierks, T. and Rescorla, E., "RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2," 2008.

[54] Dolev, D. and Yao, A., "On the Security of Public Key Protocols," *IEEE Transactions on Information Theory*, vol. 29, pp. 198–208, Mar. 1983.

[55] Eckersley, P., "A Syrian Man-In-The-Middle Attack against Facebook." `https://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook`, 2011.

[56] Eckersley, P., "How secure is HTTPS today? How often is it attacked?," 2011.

[57] Eckersley, P. and Burns, J., "An Observatory for the SSLiverse," in *DEFCON 18 Hacking Conference*, 2010.

[58] Eckersley, P. and Burns, J., "The (Decentralized) SSL Observatory," in *USENIX Security Symposium (Invited Talk)*, 2011.

[59] El Sawda, S. and Urien, P., "SIP Security Attacks and Solutions: A state-of-the-art review," in *Proceedings of the International Conference on Information and Communication Technologies (ICTTA)*, 2006.

[60] Electronic Frontier Foundation, "HTTPS Everywhere." `https://www.eff.org/https-everywhere`.

[61] Electronic Frontier Foundation (EFF), "The Sovereign Keys Project," 2011.

[62] Elizabeth Woyke, "Automatic Wi-Fi Offloading Coming To U.S. Carriers." `http://www.forbes.com/sites/elizabethwoyke/2011/04/22/automatic-wi-fi-offloading-coming-to-u-s-carriers/`, 2011.

[63] Ellison, C. and Schneier, B., "Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure," *Computer Security Journal*, vol. 16, no. 1, pp. 1–7, 2000.

[64] Engert, K., "MECAI," 2011.

[65] Engler, J., Karlof, C., Shi, E., and Song, D., "Is It Too Late for PAKE?," in *In Proceedings of the IEEE Web 2.0 Security and Privacy Workshop*, 2009.

[66] Evans, C. and Palmer, C., "Certificate Pinning Extension for HSTS," 2011.

[67] Eyers, T. and Schulzrinne, H., "Predicting Internet Telephony Call Setup Delay," in *Proc. 1st IP-Telephony Wksp*, 2000.

[68] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T., "RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1." `https://tools.ietf.org/html/rfc2616`, 1999.

[69] Firebug, "Firebug: Web Development Evolved." `https://getfirebug.com/`.

[70] Fisher, K. and Gruber, R., "PADS: Processing arbitrary data streams," in *Proceedings of Workshop on Management and Processing of Data Streams*, 2003.

[71] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L., "RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication." `https://tools.ietf.org/html/rfc2617`, 1999.

[72] Freier, A., Karlton, P., and Kocher, P., "RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0," 2011.

[73] Fu, K., Sit, E., Smith, K., and Feamster, N., "Dos and Don'ts of Client Authentication on the Web," in *Proceedings of the USENIX Security Symposium*, 2001.

[74] Galperin, E., "Microsoft Shuts off HTTPS in Hotmail for Over a Dozen Countries." `https://www.eff.org/deeplinks/2011/03/microsoft-shuts-https-hotmail-over-dozen-countries`, 2011.

[75] Gayraud, R., Jacques, O., and Wright, C. P., "SIPp: traffic generator for the SIP protocol." `http://sipp.sourceforge.net/`, 2007.

[76] GlobalSign, "Security Incident Report," 2011.

[77] Goldwasser, S., Micali, S., and Rivest., R. L., "A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988.

[78] Golwasser, S. and Micali, S., "Probabilistic Encryption," *Journal of Computer and System Sciences*, no. 28, pp. 270–299, 1984.

[79] Goodin, D., "Newfangled Cookie Attack Steals/Poisons Website Creds." http://www.theregister.co.uk/2009/11/04/website_cookie_stealing/print.html, 2009.

[80] Goodin, D., "Hotmail Always-On Crypto Breaks Microsoft's Own Apps." http://www.theregister.co.uk/2010/11/10/lame_hotmail_encryption/, 2010.

[81] Goodin, D., "Amazon Purges Account Hijacking Threat from Site." http://www.theregister.co.uk/2010/04/20/amazon_website_treat/print.html, 2011.

[82] Goodin, D., "New hack on Comodo reseller exposes private data." http://www.theregister.co.uk/2011/05/24/comodo_reseller_hacked/, 2011.

[83] Goodin, D., "RSA breach leaks data for hacking SecurID tokens." http://www.theregister.co.uk/2011/03/18/rsa_breach_leaks_securid_data/print.html, 2011.

[84] Goodin, D., "Tunisia Plants Country-Wide Keystroke Logger on Facebook," 2011.

[85] Goodin, D., "Web Authentication Authority Suffers Security Breach," 2011.

[86] Graham, R., "SideJacking with Hamster." http://erratasec.blogspot.com/2007/08/sidejacking-with-hamster_05.html, 2007.

[87] Grossman, J., "Cross-Site Tracing (XST)." http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf, 2003.

[88] Guernsey, L., "Keeping the Lifelines Open.," *The New York Times*, vol. http://www, Sept. 2001.

[89] Gustin, S., "Social Media Sparked, Accelerated Egypts Revolutionary Fire." http://www.wired.com/epicenter/2011/02/egypts-revolutionary-fire/, 2011.

[90] Gutman, P., "PKI: It's Not Dead, Just Resting," *Computer*, vol. 35, no. 8, pp. 41–49, 2002.

[91] Hachman, M., "Facebook Now Totals 901 Million Users, Profits Slip." http://www.pcmag.com/article2/0,2817,2403410,00.asp, 2012.

[92] Hamadeh, I. and Kesidis, G., "A taxonomy of internet traceback," *International Journal of Security and Networks*, vol. 1, no. 1/2, pp. 54–61, 2006.

[93] Herley, C., "So Long, and No Thanks for the Externalities," in *Workshop on New Security Paradigms Workshop (NSPW)*, 2009.

[94] Hodges, J., Jackson, C., and Barth, A., "HTTP Strict Transport Security (HSTS)." `http://tools.ietf.org/html/draft-hodges-strict-transport-sec-02`, 2010.

[95] Hoffman, P. and Schlyter, J., "IETF Internet-Draft: Using Secure DNS to Associate Certificates with Domain Names For TLS (draft-ietf-dane-protocol-06)," 2011.

[96] Holz, R. and Riedmaier, T., "Crossbear," 2012.

[97] International Organization for Standardization, "ISO/IEC 11770-4:2006: Information Technology Security Techniques Key Management Part 4: Mechanisms Based on Weak Secrets," *ISO Standard ISO/IEC*, 2006.

[98] International Telecommunication Union, "ITU-T Recommendation X.1035: Password-Authenticated Key Exchange (PAK) Protocol," 2007.

[99] International Telecommunication Union, "Recommendation ITU-T H.323: Packet-based multimedia communications systems." `http://www.itu.int/rec/T-REC-H.323-200912-I/en`, 2009.

[100] Jackson, C. and Barth, A., "Forcehttps: Protecting High-Security Web Sites from Network Attacks," in *Proceeding of the ACM International Conference on World Wide Web (WWW)*, 2008.

[101] Jackson, C., Simon, D. R., Tan, D. S., and Barth, A., "An evaluation of extended validation and picture-in-picture phishing attacks," in *Proceedings of the International Conference on Financial cryptography*, 2007.

[102] Janak, J., "SIP Proxy Server Effectiveness," *Master's Thesis, Department of Computer Science, Czech Technical University, Prague, Czech*, 2003.

[103] Jehiah, "XSS - Stealing Cookies 101." `http://jehiah.cz/a/xss-stealing-cookies-101`, 2006.

[104] Juels, A., Jakobsson, M., and Jagatic, T., "Cache Cookies for Browser Authentication (Extended Abstract)," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[105] Kaliski, B., "PKCS 5: Password-Based Cryptography Specification Version 2.0." `http://tools.ietf.org/html/rfc2898`, 2000.

[106] Kaufman, C., Perlman, R., and Speciner, M., *Network Security: Private Communication in a Public World*. Prentice Hall, second ed., 2002.

[107] Keizer, G., "Hackers May Have Stolen Over 200 SSL Certificates," 2011.

[108] Kim, J., Biryukov, A., Preneel, B., and Hong, S., "On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (extended abstract)," in *Proceedings of the International Conference on Security and Cryptography for Networks*, 2006.

[109] Kirk, J., "KPN Stops Issuing SSL Certificates After Possible Breach," 2011.

[110] Koch, A., "DroidSheep." `http://droidsheep.de/`, 2011.

[111] Kolsek, M., "Session Fixation Vulnerability in Web-based Applications." `http://www.acrossecurity.com/papers/session_fixation.pdf`, 2007.

[112] Krawczyk, H., Bellare, M., and Canetti, R., "RFC 2104 - HMAC: Keyed-Hashing for Message Authentication." `http://www.ietf.org/rfc/rfc2104.txt`, 1997.

[113] Kristol, D. and Montulli, L., "RFC 2109 - HTTP State Management Mechanism." http://tools.ietf.org/html/rfc2109, 1997.

[114] Kristol, D. and Montulli, L., "RFC 2965 - HTTP State Management Mechanism," 2000.

[115] Lamport, L., "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[116] Langley, A., "Overclocking SSL." `http://www.imperialviolet.org/2010/06/25/overclocking-ssl.html`, 2010.

[117] Langley, A., "Revocation Doesn't Work," 2011.

[118] Laurie, B. and Langley, A., "Certificate Authority Transparency and Auditability," 2011.

[119] Lemos, R., "Cyber Attacks Disrupt Kyrgyzstan's Networks." `http://www.securityfocus.com/brief/896`, Jan. 2009.

[120] Leyden, J., "AmEx 'Debug Mode Left Site Wide Open', Says Hacker." `http://www.theregister.co.uk/2011/10/07/amex_website_security_snafu/print.html`, 2011.

[121] Leyden, J., "Inside 'Operation Black Tulip': DigiNotar Hack Analysed," 2011.

[122] Leyden, J., "Trustwave Admits Crafting SSL Snooping Certificate," 2012.

[123] Liu, A., Kovacs, J., and Gouda, M., "A Secure Cookie Protocol," in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 2005.

[124] Liu, D. and Ning, P., "Multilevel μTESLA: Broadcast authentication for distributed sensor networks," *ACM Transactions in Embedded Computing Systems (TECS)*, vol. 3, no. 4, pp. 1539–9087, 2004.

[125] Liu, H. H., "Applying Queuing Theory to Optimizing the Performance of Enterprise Software Applications," in *Proceedings of the Computer Measurement Groups International Conference*, 2006.

[126] MacKenzie, P., "The PAK suite: Protocols for Password-Authenticated Key Exchange," in *IEEE P1363.2: Password-Based Public-Key Cryptography*, 2002.

[127] Mackenzie, P. and Patel, S., "Hard Bits of the Discrete Log with Applications to Password Authentication," in *Topics in Cryptology CT-RSA 2005*, vol. 3376 of *Lecture Notes in Computer Science*, pp. 209–226, Springer Berlin / Heidelberg, 2005.

[128] Marlinspike, M., "SSLStrip." `http://www.thoughtcrime.org/software/sslstrip/`, 2009.

[129] Marlinspike, M., "Convergence," 2011.

[130] Meenakshi, S. and Raghavan, S., "Impact of IPSec Overhead on Web Application Servers," in *Proceedings of the International Conference on Advanced Computing and Communications (ADCOM)*, 2006.

[131] Menn, J., "Key Internet Operator VeriSign Hit by Hackers," 2012.

[132] Microsoft, "Windows Live ID." `http://www.passport.net/`, 2012.

[133] Miller, R., "Facebook Server Count: 60,000 or More." `https://www.datacenterknowledge.com/archives/2010/06/28/facebook-server-count-60000-or-more/`, 2010.

[134] Miltchev, S., Ioannidis, S., and Keromytis, A. D., "A Study of the Relative Costs of Network Security Protocols," in *Proceedings of the FREENIX Track: USENIX Annual Technical Conference (ATC)*, 2002.

[135] Mitchell, C. J. and Chen, L., *Comments on the S/KEY user authentication scheme*, vol. 30. New York, NY, USA: ACM, 1996.

[136] Mitchell, S., "Understanding ASP.NET View State." `http://msdn.microsoft.com/en-us/library/ms972976.aspx`, 2004.

[137] Mosberger, D. and Jin, T., "httperf - A Tool for Measuring Web Server Performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[138] M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., and Ranen, O., "RFC 4226 - HOTP: An HMAC-Based One-Time Password Algorithm." `http://tools.ietf.org/html/rfc4226`, 2005.

[139] Mukerjee, S., "Subscriber Management For Next-Generation Networks." `http://www.xchangemag.com/webexclusives/62h2815505.html`, 2006.

[140] MySQL, "The World's Most Popular Open Source Database." http://www.mysql.com/, 2009.

[141] Nahum, E. M., Tracey, J., and Wright, C. P., "Evaluating SIP server performance," in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[142] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "RFC 4120 - The Kerberos Network Authentication Service (V5)." https://tools.ietf.org/html/rfc4120, 2005.

[143] Oiwa, Y., Watanabe, H., Takagi, H., Ioku, Y., and Hayashi, T., "Mutual Authentication Protocol for HTTP," 2011.

[144] Oiwa, Y., Takagi, H., Watanabe, H., and Suzuki, H., "PAKE-based Mutual HTTP Authentication for Preventing Phishing Attacks (Poster)," in *Proceedings of the International Conference on World Wide Web (WWW)*, 2009.

[145] Ono, K. and Schulzrinne, H., "One Server Per City: Using TCP for Very Large SIP Servers," in *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm)*, 2008.

[146] OpenID, "OpenID Foundation Website." http://openid.net/, 2012.

[147] OpenSER, "OpenSER - the Open Source SIP Server." http://www.opensips.org/.

[148] Oppliger, R., Hauser, R., and Basin, D., "SSL/TLS Session-Aware User Authentication," *Computer*, vol. 41, no. 3, pp. 59–65, 2008.

[149] Oracle, "Voicemail & Fax Administrator's Guide 10g, Release 1." http://download.oracle.com/docs/cd/b1559501/mail.101/b14496.pdf, 2005.

[150] Park, J. S. and Sandhu, R., "Secure Cookies on the Web," *IEEE Internet Computing*, vol. 4, pp. 36–44, 2000.

[151] Parno, B., Kuo, C., and Perrig, A., "Phoolproof Phishing Prevention," in *Proceedings of Financial Cryptography and Data Security*, 2006.

[152] Perrig, A., Szewczyk, R., Wen, V., Culler, D., and Tygar, J. D., "SPINS: Security Protocols for Sensor Networks," in *Proceedings of the International Conference on Mobile Computing and Networks (MOBICOM)*, 2001.

[153] Ponurkiewicz, B., "FaceNiff." http://faceniff.ponury.net/, 2011.

[154] Prandini, M., Ramilli, M., Cerroni, W., and Callegati, F., "Splitting the HTTPS Stream to Attack Secure Web Connections," *IEEE Security and Privacy*, vol. 8, pp. 80–84, 2010.

[155] Prince, B., "Google Moves Encrypted Web Search." http://www.eweek.com/c/a/Security/Google-Moves-Encrypted-Web-Search-668624/, 2010.

[156] Rashid, F. Y., "IE Flaw Lets Attackers Steal Cookies, Access User Accounts." http://www.eweek.com/c/a/Security/IE-Flaw-Lets-Attackers-Steal-Cookies-Access-User-Accounts-402503/, 2011.

[157] Reis, C., Gribble, S. D., Kohno, T., and Weaver, N. C., "Detecting In-Flight Page Changes with Web Tripwires," in *Proceedings of the USENIX Symposium on Network Systems Design and Implementation (NSDI)*, 2008.

[158] Richmond, R., "An Attack Sheds Light on Internet Security Holes," 2011.

[159] Richtel, M., "Inauguration Crowd Will Test Cellphone Networks," *The New York Times*, vol. http://www, 2009.

[160] Ristic, I., "Internet SSL Survey 2010," 2010.

[161] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E., "RFC 3261 - SIP: Session Initiation Protocol." https://tools.ietf.org/html/rfc3261, 2002.

[162] Salsano, S., Veltri, L., and Papalilo, D., "SIP security issues: the SIP authentication procedure and its processing load," *IEEE Network*, vol. 16, no. 6, pp. 38–44, 2002.

[163] Savage, S., Wetherall, D., Karlin, A., and Anderson, T., "Practical network support for IP traceback," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2000.

[164] Schechter, S. E., Dhamija, R., Ozment, A., and Fischer, I., "The Emperor's New Security Indicators," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2007.

[165] Schneier, B., "Man-in-the-Middle Attack Against SSL 3.0/TLS 1.0." https://www.schneier.com/blog/archives/2011/09/man-in-the-midd_4.html, 2011.

[166] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V., "RFC 3550 - RTP: A Transport Protocol for Real-Time Applications." https://tools.ietf.org/html/rfc3550, 2003.

[167] Schulzrinne, H., Narayanan, S., Lennox, J., and Doyle, M., "SIPstone-Benchmarking SIP Server Performance," in *Columbia University Technical Report*, 2002.

[168] Shacham, H. and Boneh, D., "Improving SSL Handshake Performance via Batching," in *Proceedings of the Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, 2001.

[169] Shen, C., Schulzrinne, H., and Nahum, E., "Session Initiation Protocol (SIP) Server Overload Control: Design and Evaluation," in *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm)*, 2008.

[170] Shoup, R. and Pritchett, D., "The eBay Architecture." `https://encrypted.google.com/url?sa=t&rct=j&q=the%20ebay%20architecture&source=web&cd=1&ved=0CGoQFjAA&url=http%3A%2F%2Fwww.addsimplicity.com%2Fdownloads%2FeBaySDForum2006-11-29.pdf&ei=QUSlT_7wN4Ko8QSfuOmXAw&usg=AFQjCNF04TT6k3T8qXGaQpo_isPbvO_Fpg&cad=`, 2006.

[171] Siegler, M., "China Syndrome: Gmail Now Defaults To Encrypted Access." `http://techcrunch.com/2010/01/13/china-hacking-gmail-secure/`, 2010.

[172] Singel, R., "Law Enforcement Appliance Subverts SSL," 2010.

[173] Singh, K. and Schulzrinne, H., "Failover and load sharing in SIP telephony," *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Philadelphia, PA, July*, 2005.

[174] Singh, K., Schulzrinne, H., and Lennox, J., "SIP Server Scalability," 2005.

[175] Singh, K., *Reliable, Scalable and Interoperable Internet Telephony*. PhD thesis, Columbia University, 2006.

[176] Soghoian, C. and Stamm, S., "Certified Lies: Detecting and Defeating Government Interception Attacks Against SSL," in *Proceedings of Financial Cryptography and Data Security*, 2011.

[177] Song, D. and Perrig, A., "Advanced and Authenticated Marking Schemes for IP Traceback," in *In Proceedings of IEEE Infocom*, 2001.

[178] Stone-Gross, B., Sigal, D., Cohn, R., Morse, J., Almeroth, K., and Kruegel, C., "VeriKey: A Dynamic Certificate Verification System for Public Key Exchanges," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2008.

[179] Stuttard, D. and Pinto, M., *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. Wiley, 2 ed., 2011.

[180] Sunshine, J., Egelman, S., Almuhimedi, H., and Atri, N., "Crying Wolf: an Empirical Study of SSL Warning Effectiveness," in *Proceedings of the Usenix Security Symposium*, 2009.

[181] Sweeney, A., Doucette, D., Hu, W., Anderson, C., Nishimoto, M., and Peck, G., "Scalability in the XFS file system," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 1996.

[182] Syamsuddin, I., Dillon, T., Chang, E., and Han, S., "A Survey of RFID Authentication Protocols Based on Hash-Chain Method," in *Proceedings of the International Conference on Convergence and Hybrid Information Technology (ICCIT)*, 2008.

[183] Tao, C., Qiang, G., and Baohong, H., "A lightweight authentication scheme for Session Initiation Protocol," in *Proceedings of the IEEE International Conference on Communications, Circuits and Systems (ICCCAS)*, pp. 502–505, 2008.

[184] Taylor, D., Wu, T., Mavrogiannopoulos, N., and Perrin, T., "RFC 5054 - Using the Secure Remote Password (SRP) Protocol for TLS Authentication," 2007.

[185] The CA / Browser Forum, "Baseline Requirements for the Issuance and Management of Publicly Trusted Certificates." `http://www.cabforum.org/Baseline_Requirements_V1.pdf`, 2011.

[186] The Open Web Application Security Project (OWASP), "Cross-site Scripting (XSS)." http://www.owasp.org/index.php/Cross-site_Scripting_

[187] The Open Web Application Security Project (OWASP), "OWASP Top Ten Project." `http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`, 2010.

[188] Towns, S., "American Idols Impact." `http://www.govtech.com/featured/Worlds-Largest-Telcom-Manages-Network-PHOTOSVIDEO.html#idol`, 2012.

[189] Traynor, P., Lin, M., Ongtang, M., Rao, V., Jaeger, T., La Porta, T., and Mcdaniel, P., "On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core," *CCS '09: Proceedings of the 15th ACM conference on Computer and communications security*, 2009.

[190] Turner, S. and Chen, L., "RFC 6151 - Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms." `http://tools.ietf.org/html/rfc6151`, 2011.

[191] Visaggio, C., "Session Management Vulnerabilities in Today's Web," *IEEE Security and Privacy*, vol. 8, pp. 48–56, 2010.

[192] Wagoner, L., *Detecting Man-in-the-Middle Attacks against Transport Layer Security Connections with Timing Analysis*. PhD thesis, Air Force Institute of Technology, 2011.

[193] Wang, C.-H. and Liu, Y.-S., "A dependable privacy protection for end-to-end VoIP via Elliptic-Curve Diffie-Hellman and dynamic key changes," *Journal of Network and Computer Applications*, vol. 34, no. 5, pp. 1545–1556, 2011.

[194] Wang, X., Yin, Y. L., and Yu, H., "Finding collisions in the full SHA-1.," in *Proceedings of Crypto*, 2005.

[195] Wang, X. and Yu, H., "How to Break MD5 and Other Hash Functions," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2005.

[196] Wendlandt, D., Andersen, D. G., and Perrig, A., "Perspectives: Improving SSH-style Host Authentication with Multi-path Probing," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2008.

[197] Williams, N., "RFC 5056 - On the Use of Channel Bindings to Secure Channels," 2007.

[198] WordPress, "WordPress: Blog Tool, Publishing Platform, and CMS." `http://wordpress.org/`.

[199] Wu, T., "The Secure Remote Password Protocol," in *Proceedings of the Network and Distributed System Security Symposium*, 1998.

[200] Wyke, J., "What is Zeus?." `http://www.sophos.com/medialibrary/PDFs/technical%20papers/Sophos%20what%20is%20zeus%20tp.pdf`, 2011.

[201] Xenitellis, S., "Certificate Watch Firefox Security Add-on," 2010.

[202] Xia, H. and Brustoloni, J. C., "Hardening Web Browsers Against Man-in-the-Middle and Eavesdropping Attacks," in *Proceedings of the International Conference on World Wide Web (WWW)*, 2005.

[203] Yang, C.-C., Wang, R.-C., and Liu, W.-T., "Secure authentication scheme for session initiation protocol," *Computers & Security*, vol. 24, no. 5, pp. 381–386, 2005.

[204] Ye, Z. E. and Smith, S., "Trusted Paths for Browsers," in *Proceedings of the Usenix Security Symposium*, 2002.

[205] Yoon, E.-J. and Yoo, K.-Y., "A New Authentication Scheme for Session Initiation Protocol," in *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems*, 2009.

[206] Zalewski, M., "Browser Security Handbook." `http://code.google.com/p/browsersec/wiki/Part2`, 2008.

[207] Zhang, R., Wang, X., Yang, X., and Jiang, X., "Billing attacks on SIP-based VoIP systems," in *Proceedings of the first USENIX Workshop on Offensive Technologies*, USENIX Association, 2007.

[208] Zhou, Y. and Evans, D., "Why Aren't HTTP-only Cookies More Widely Deployed?," in *Proceedings of the Web 2.0 Security and Privacy Workshop (W2SP)*, 2010.