# Layering Security at Global Control Points to Secure Unmodified Software

Scott Ruoti
MIT Lincoln Laboratory
*scott@ruoti.org*

Kent Seamons
Brigham Young University
*seamons@cs.byu.edu*

Daniel Zappala
Brigham Young University
*zappala@cs.byu.edu*

***Abstract*—Developing secure software is inherently difficult, and is further hampered by a rush to market, the lack of cybersecurity-trained architects and developers, and the difficulty of identifying flaws and deploying mitigations. To address these problems, we advocate for an alternative paradigm—layering security onto applications from global control points, such as the browser, operating system, or network. This approach adds security to existing applications, relieving developers of this burden. The benefits of this paradigm are three-fold—(1) increased correctness in the implementation of security features, (2) coverage for all software, even non-maintained legacy software, and (3) more rapid and consistent deployment of threat mitigations and new security features. To demonstrate these benefits, we describe three concrete instantiations of this paradigm—MessageGuard, a system that layers end-to-end encryption in the browser; TrustBase, a system that layers authentication in the operating system; and software-defined perimeter, which layers access control at network middleboxes.**

## I. The Case for an Alternate Paradigm

Many of today's software products are insecure. Often, security is ignored as companies race to be first-to-market. Subsequent attempts to bolt security onto existing products face many challenges, frequently leaving residual vulnerabilities. The current paradigm for developing secure software is failing, and it is essential to explore alternatives.

To make progress, it is important to first understand the drawbacks of the current secure software development paradigm—i.e., implementing security on an application-by-application basis. In this model, each application needs to be architected with security in mind, and many—if not most—application developers must then correctly implement the relevant security features. Unfortunately, there is a significant lack of developers trained in cybersecurity [1], meaning that the architecture and implementation are both likely to have security flaws. Moreover, there is no indication that the number of cybersecurity-trained developers will ever scale up sufficiently to support the ever-increasing need for new applications.

To partially address this problem, security-focused software libraries (e.g., OpenSSL, BouncyCastle, PyCrypto) have been developed to provide trustworthy instantiations of basic cryptographic primitives and protocols. While these libraries can help developers implement specific security functionality, they require extensive knowledge to use correctly [2], with experience showing that thousands of applications are broken even when performing what should be a relatively simple task of verifying website authenticity [3], [4], [5], [6]. Additionally, when vulnerabilities are eventually found in these security libraries (e.g., Heartbleed in OpenSSL), it is hard to ensure the deployment of mitigations to all affected software. Not only do developers need to become aware of the flaw, integrate the mitigation, and deploy the updated software, but in many cases, users must also be aware of the problem and proactively update all of their compromised software.

Finally, even if applications are properly architected and implemented, it is likely that they will one day become legacy applications and cease to be supported. After support ends, even properly implemented applications can see flaws emerge as the underlying operating system and runtimes change. Moreover, the lack of ongoing support means these flaws are unlikely to be addressed. The fact that legacy software is frequently used not only directly by users, but also as subsystems to other applications only further compounds the challenge.

## II. The Layered Security Paradigm

Regardless of the specific reason, the result of the current paradigm is thousands, if not tens-of-thousands, of applications with broken and outdated security. To address these issues, we advocate for an alternative paradigm—instead of implementing security into individual applications, it should instead be implemented at global control points, which are then responsible for layering security on top of the applications. For simplicity, in this paper we refer to this paradigm as the *layered security* paradigm.

The workflow for this paradigm is as follows (see Figure 1):

1) The application initiates an insecure action that needs additional security. For example, requesting a website over HTTP, authenticating using a password (non-two-factor authentication), or requesting a plaintext file).
2) The global control point intercepts the insecure action and layers the appropriate security by requesting a secure action. For example, accessing the website using HTTPS, authenticating using two-factor authentication, or requesting an encrypted file. The modified action then takes place.
3) The global control point intercepts the result and modifies it to make it suitable for the application. For example, returning the result of the password authentication or decrypting a file before returning it.
4) The application receives a response with the correct information.

The layered security paradigm removes the need for developers of individual applications to be cybersecurity experts,

978-1-5386-3467-7/17 $31.00 © 2017 IEEE
DOI 10.1109/SecDev.2017.20

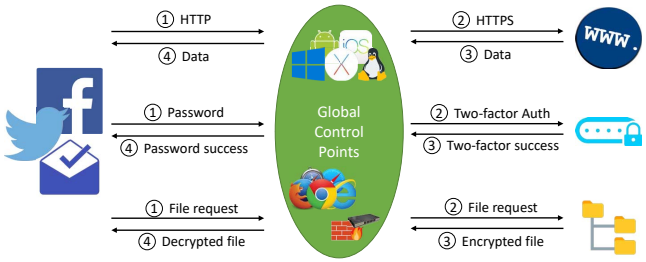



Fig. 1. Alternate Paradigm Flow Diagram.

instead moving this burden to the developers of the global control points (e.g., browsers, operating systems, firewalls). As the number of global control point implementations is many orders of magnitude smaller than the number of applications, this paradigm sidesteps the largest limitation of the existing secure software development paradigm. Application developers will continue to develop applications as they do today, largely agnostic of security, and as needed the global control point will automatically and transparently layer the appropriate security functionality. Critically, the new security functionality is added without the knowledge or control of the underlying application. This means that all applications—including existing applications—are supported by the layered security paradigm, regardless of whether they were designed with it in mind.

We note that this is not the first paper that describes the layered security approach. In our own work [7], [8], [9] and that of others [11], [12], [13], [14], [10], this paradigm has been used to secure specific classes of applications (e.g., email) and operations (e.g., HTTPS). Still, to our knowledge, this is the first paper that abstracts the layered security paradigm from a specific application area and discusses how it could be applied more generally as an alternative to the failing existing secure software development paradigm.

### A. Benefits

The benefits of the layered security paradigm are three-fold—correctness, coverage, and speed of deploying updates.

First, this paradigm increases the correctness of deployed security functionality. Instead of needing to correctly implement security in thousands of applications, it is only necessary to correctly implement security at the global control points. The smaller number of global control points means that they can receive increased scrutiny by the limited number of cybersecurity-trained architects and developers. Many global control points like browsers and operating system are already maintained by large organizations, and these companies—for example, Google and Microsoft—are well-equipped to handle this type of work.

Second, it increases the coverage of security features in applications. The global control point will layer security on to *all* applications, providing protection even to applications that were not designed with security in mind, have incorrect implementations, and are not being actively maintained.

Third, it increases the speed at which updates can be deployed. Instead of needing to update each application individually, new features and threat mitigations can be deployed in a single update that impacts all applications. Having fewer but well-maintained systems providing security results in updates that address vulnerabilities everywhere they are relevant.

### B. Limitations

Layering security at a global control point has several limitations. First, it introduces a single point of failure that affects all applications. Although this presents a significant risk, we believe it is preferable to the current state of application security. It is better to have a well-maintained, difficult-to-compromise single point of failure than thousands of poorly maintained applications with known vulnerabilities. Second, layering security onto an unaware application has the potential to disrupt that application's functionality and usability. Extreme care must be taken when developing the layered security system to avoid this problem.

### C. Potential Global Control Points

The feasibility of this paradigm rests on finding global control points where security can be added without cooperation from applications. We identify three potential global control points, though there are certainly others possible:

- *The browser* is a natural global control point for web applications. The web has become the de facto method for deploying new applications, acting as the central point in the Internet's architecture [15]. Some work has sought to exploit this prominence to design new browser architectures with strong security properties [16], [17], [39]. Within the browser, there are many places to layer security including when creating and rendering the DOM, as well as when processing network traffic.
- *The operating system* acts as a good global control point for applications running on a single device. Firewalls and anti-virus programs have long operated at this global control point to impose security policies on applications. The operating system has such broad coverage that there is little limit to the locations security can be layered here— e.g., at the file system, at the network layer, before content is rendered.
- *Network middleboxes* is a common global control point for an organization. These middleboxes are already used to layer traffic filtering and threat detection on top of existing network applications. There are many interesting potential uses for these middleboxes, including enforcing proper TLS authentication or requiring two-factor authentication for legacy applications.

While the layered security paradigm clearly is a natural fit for network security, the approach generalizes more broadly. For example, an operating system control point could monitor allocated memory and prevent buffer overflows. Likewise, an operating system global control point could augment local applications with safe password entry and strong password protocols [18].

### D. Layering-Aware Applications

While this paradigm works with unmodified applications, it can be even more powerful for layering-aware applications. These applications can share context-specific information regarding the operation to the global control point. While the global control point ultimately decides what action to take, this context can allow the global control point to make more informed decisions about what action to take. Additionally, layering-aware applications can avoid implementing security features they know will be provided by the global control point.

A good example of this approach is adding end-to-end encryption into web applications. A variety of web applications could use this functionality, such as direct messaging in Facebook and Twitter, or shared document editing such as Google Docs, so that sensitive information is hidden from service providers. If these applications could provide plaintext and a user identifier, the global control point could then encrypt the plaintext for the application, ensuring confidentiality and integrity.

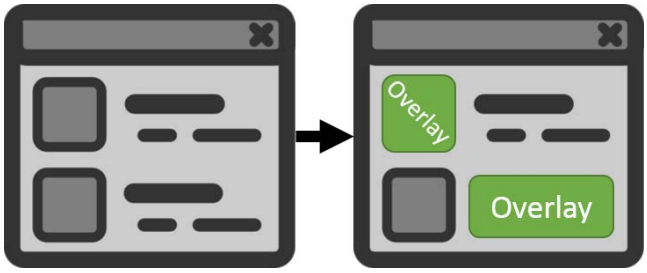
## III. Example Use Cases

To demonstrate the feasibility of the layered security paradigm, we consider the three global control points mentioned above—the browser, the operating system, and network middleboxes. We first describe MessageGuard, a system from our own research that uses a browser global control point to layer end-to-end encryption into web applications. We then describe TrustBase, a system we developed that uses an operating system global control point to provide system administrator control over certificate-based authentication of remote servers. Finally, we describe the use of a software-defined perimeter, a technique from industry that layers additional access control onto network services from network middleboxes.

### A. MessageGuard

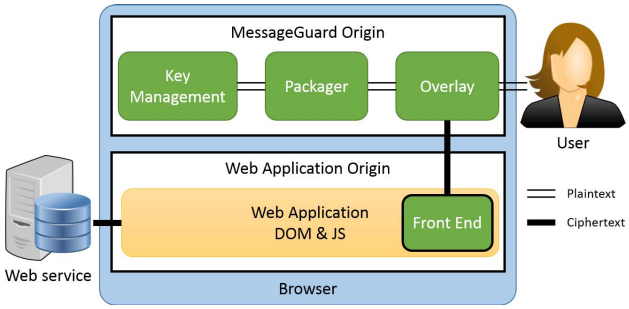
Users share private information on the web through a variety of applications, such as email, instant messaging, social media, and document sharing. HTTPS protects this information during transmission but does not protect users' data while at rest. Additionally, middleboxes can weaken HTTPS connections by failing to properly implement TLS or adequately validate certificate chains [19]. Even if a website correctly employs HTTPS and encrypts user data while at rest, the user's data is still vulnerable to honest-but-curious data mining [7], third-party library misbehavior [20], website hacking, protocol attacks [21], [22], [23], and government subpoena.

This state of affairs motivates the need for end-to-end encryption of user data– a user's sensitive data is encrypted at their computer and only decrypted once it reaches the intended recipient, remaining opaque to the applications (e.g., websites) that store or transmit this encrypted data. Instead of trying to retrofit the multitude of applications that could benefit from end-to-end encryption, we have built MessageGuard [7], [8], [9], a system that layers end-to-end encryption on top of existing web applications. MessageGuard uses the browser as its global control point and is deployed as either a browser extension or a bookmarklet.




Portions of the web application are replaced with security overlays.

Fig. 2. Security overlays.




User's sensitive data is only accessible within the MessageGuard origin.

Fig. 3. Overview of the MessageGuard architecture.

*1) Design:* MessageGuard uses *security overlays* [7] implemented as iFrames [8] to layer end-to-end encryption into existing web applications (see Figure 2). Inside a security overlay, users interact with the plaintext contents of their sensitive data, while the underlying applications only have access to the end-to-end encrypted ciphertext. MessageGuard's overlays are designed to be functionally transparent to users, allowing users to complete tasks as they are accustomed to, while still protecting the data from the web application.

MessageGuard's architecture is given in Figure 3. The *front end component* is the bridge between MessageGuard and the underlying application. It is responsible for scraping encrypted text from the web application and sending it to the overlay. Similarly, when text is authored in the overlay, the ciphertext is passed to the front end and inserted into the web application. Importantly, the front end component has no direct access to the overlay's content and is never given access to an overlay's sensitive data. MessageGuard includes a general front end that works with most applications but also supports application-specific front ends for greater integration and usability.

Within an overlay, a substitute interface is displayed to allow users to view and author sensitive data. These overlays user a *packager* to wrap encrypted data in a form suitable for transmission to the front end and eventual insertion into the underlying web application. The packager is also responsible for reversing this process when an encrypted package is received from the front end. Finally, the *key management* component is responsible for storing and using the user's keys. MessageGuard includes general implementations for a read overlay, a write overlay, a packager; we also include key management components for PGP, IBE, and a password-based key derivation.

MessageGuard is able to add end-to-end encryption to any application that displays data to users or allows the user to enter data. Critically, this is done without requiring the underlying application to be modified. This is important to allow for the rapid deployment of end-to-end encryption, regardless of the number of disparate systems that would otherwise need to be updated (e.g., email). Still, MessageGuard can benefit from MessageGuard-aware clients. These clients can provide contextual clues to the MessageGuard front end—for example, the type of interface to display (e.g., a short-form or long-form composition interface) or content size limits—enabling tighter and more consistent integration with MessageGuard. Alternatively, applications could provide application-specific overlays, exactly replicating the expected functionality of the application in the overlay. This is far easier than adding end-to-end encryption directly to the application because it abstracts encryption standards and key management away from application developers.

*2) Ubiquity and Performance:* We tested MessageGuard on major browsers and it worked in all cases: Desktop—Chrome, Firefox, Internet Explorer, Opera, and Safari. Android—Chrome, Firefox, Opera. iOS—Chrome, Mercury, Safari. We also tested MessageGuard on the Alexa top 50 web sites. One of the sites is not a web application (t.co), and another requires a Chinese phone number to use it (weibo.com). MessageGuard was able to encrypt data in 47 of the 48 remaining web applications. The one site that failed (youtube.com) did so because the application removed the comments field when it lost focus, which happens when focus switched to MessageGuard's compose overlay. We were able to address this problem with a customized front end that required only five lines of code to implement.

We profiled MessageGuard on several popular web applications and analyzed MessageGuard's impact on load times. In each case, we started the profiler, reloaded the page, and stopped profiling once the page was loaded. Our results show that MessageGuard has little impact on page load times and does not degrade the user's experience as they surf the Web: Facebook – 0.93%, Gmail – 2.92%, Disqus – 0.54%, Twitter – 1.98%.

Since MessageGuard is intended to work with all websites, we created a synthetic web app that allowed us to test MessageGuard's performance under extreme load. This app measures MessageGuard's performance when overlaying static content present at page load (Stage 1) and when overlaying dynamic content that is added to the page after load (Stage 2). Using this synthetic web application, we tested MessageGuard with six browsers, finding that overhead grows linearly for Stage 1 content, and super-linearly for Stage 2 content. Further analysis showed that the super-linear growth of Stage 2 content was due to inefficient browser implementations and that it could grow linearly as browsers improve their implementations—as is already the case in Firefox. Regardless, even in extreme cases (Stage 2, 1,000 overlays) overlaying occurs quickly (max 61 ms).

*3) Usability:* To validate the usability of overlaying end-to-end encryption on top of existing applications, we conducted a series of IRB-approved usability studies.

First, we evaluated Private Facebook Chat [24], a system overlaying end-to-end encryption on top of Facebook Chat. Almost all users were able to use MessageGuard to encrypt their chat sessions, except two extremely novice users that were unable to complete any tasks. Additionally, users indicated that they were generally satisfied with PFC and that they would be interested in using it in practice.

Second, we evaluated Private WebMail (Pwm), a system for overlaying end-to-end encryption on top of GMail's web application. Later, we expanded Pwm to support non-email applications, and at this point we renamed it MessageGuard [8]. We evaluated Pwm across five different IRB-approved usability studies, including a total of 186 participants. Each of these studies utilized a standard usability metric, the System Usability Scale (SUS) [25], [26], to compare Pwm against prior versions and alternative approaches.

The first study (25 participants) had participants install Pwm using a bookmarklet, and the second (32 participants) and third study (28 participants) had participants install Pwm using an extension [7]. This early version of Pwm averaged a SUS score of 73.8, putting it in the 70$^{th}$ percentile of systems tested with SUS — first study (75.7), second study (70.7), and third study (70.7). All future studies used the extension installation method.

Based on feedback from the first three studies, we further refined Pwm's design and tested it in a fourth study (51 participants) [9]. The results of this test was a SUS score of 80.0, falling in the 88$^{th}$ percentile of systems tested with SUS, and nearly all participants (92%) believed that their friends and family could easily start using Pwm. In a fifth study (fifty participants) examining whether Pwm could be adopted in a grassroots fashion [27], [28], Pwm received a SUS score of 72.3, falling in the 63$^{rd}$ percentile of systems tested with SUS.[1]

In the first, second, third, and fifth studies we also compared Pwm against alternative secure email systems—Encipher.it, Mailvelope, MessageProtector, Tutanota, Virtru, and Voltage Mail. In each case, Pwm was always rated as the most usable system. This demonstrates that overlaying user-visible security can be just as usable as integrating that security directly into the application.

### B. TrustBase

TrustBase was designed to address significant flaws in certificate-based authentication of remote servers [10]. Authentication of the server is critical part of TLS, yet many applications do not properly validate the server's certificate [3], [4], [5], [6]. In addition, the Certificate Authority (CA) system itself is vulnerable because most CAs can create certificates for any host. When DigiNotar was hacked in 2011, the perpetrators were able to create hundreds of falsified certificates that were accepted by all browsers [29]. CAs have been shown to be vulnerable to a variety of weaknesses [30], [31], and governmental ownership or access creates the possibility of malfeasance [32], [33]..

To address these problems, TrustBase uses an operating system control point to give system administrators and OS

[1]In this study we also examined other systems, all of which received uncharacteristically low scores, suggesting that Pwm's true SUS score is much closer to the 80 found in our fourth study. We are currently replicating this study to correct for experimental bias. Initial results have put Pwm's SUS score back around 80.

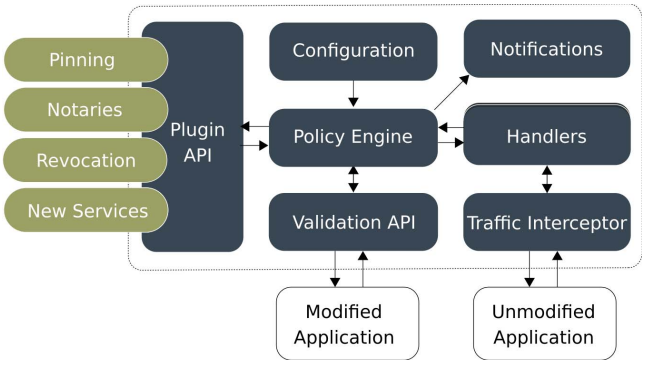



Fig. 4. TrustBase architecture.

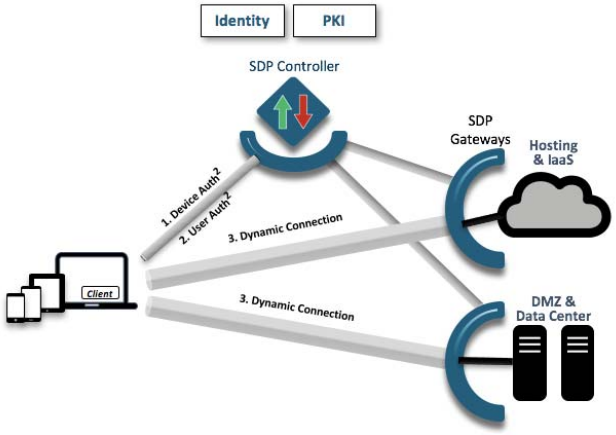



Fig. 5. Architecture for software defined perimeter (SDP).

vendors control over authentication policy. The architecture for TrustBase is shown in Figure 4. The *traffic interceptor* and a set of *handlers* operate in the kernel to intercept all outgoing traffic and identify certificates that should be validated, such as in a TLS handshake. The certificates are given to a userspace *policy engine*, which coordinates with a variety of *authentication systems* to validate the certificates, enforcing whatever aggregation policy the system administrator chooses. An evaluation of TrustBase shows that it has negligible overhead.

The use of an operating system global control point affords TrustBase some interesting advantages. Because TrustBase intercepts traffic at the socket layer, before traffic is delivered to the transport protocol, it is able to secure all applications, without modification. This is particularly important given the large number of applications that have been found to improperly implement certificate authentication. Moreover, the modular architecture enables TrustBase to be used to control authentication for a variety of protocols; currently TrustBase supports TLS and STARTTLS, but it can easily be extended to QUIC and DTLS, for example.

TrustBase also demonstrates the utility of layering-aware applications. The architecture includes an API so that modified applications can directly ask the policy engine for authentication by providing a hostname, certificate, and relevant metadata. Calling this API enables the application to avoid implementing authentication, which would eliminate needless duplication of functionality that the operating system is providing.

The generality of TrustBase is demonstrated by virtue of implementations on Linux, Windows, and Android, as well as a wide range of authentication systems that have been developed. A certificate revocation plugin extends OCSP checking to all applications. A CRLSet blocking plugin checks Google's CRLSet to determine whether a certificate should be blocked, extending Chrome's protection to all applications. A DANE plugin [34] uses DNS to distribute public keys that should be used to sign certificates for a domain. A notary plugin provides multi-path probing to check certificates, similar to ideas promoted by Perspectives [35] and Convergence [30]. The plugin architecture also enables some novel enhancements beyond certificate authentication—one security service enforces a secure default configuration for TLS and disables insecure cipher suites.

### C. Software-Defined Perimeter (SDP)

Many organizations protect their network services by placing them inside the organization's local area network (LAN). Only computers connected to the LAN have access to these services. If a user outside the LAN needs access to these services, they can initialize a virtual private network (VPN) connection to the LAN and access the services. The problem with this approach is that once an adversary gains access to the LAN (either physical or remote), the attacker can move laterally, compromising other machines and services on the network with relative ease.

One approach to addressing this problem is a software defined perimeter (SDP) [36], [37]. With SDP, all of an organization's network services are initially dark, refusing to respond to any network packets—to an attacker they might well not even exist. For a client to talk to a network service, they must first register their intent with the SDP Controller. Once approved, the SDP Controller gives the client an access token that they can present to a dark network service—or potentially a gateway that will tunnel connections to those services—to begin communicating with that network service. Unlike a VPN, SDP does not grant blanket access to the network, but rather customizes user tokens to only allow access to the services that the user should have access to. This effectively layers fine-grained access control on top of the organization's network.

Once SDP has been deployed, the SDP controller becomes a global control point at which additional access control can be layered onto network services. For example, the SDP controller can require that users authenticate to the SDP controller using multi-factor authentication, effectively layering multi-factor authentication on top of all the organization's network services. Alternatively, the SDP controller can layer role-based access control (RBAC) or attributed-based access control (ABAC) on top of network services by only providing tokens for services for which the user has the correct roles or attributes. Other examples of security that can be layered onto network services include asset management (only authorized devices can access the services), patch management (only patched devices can access the service), and access revocation (immediate termination of access for a compromised client).

Critically, this layering happens without modifying the underlying network services. While many new applications have begun deploying some of the above features (e.g., multi-factor authentication), many more do not. While these features may slowly percolate to some applications, there is a significant body of legacy software that is no longer maintained. This is important, as some legacy systems are currently completely unsecured (open to all on the LAN) or protected only by a trivially-compromised mechanism (single hard-coded password).

In the current secure software development paradigm, *all* legacy applications would need to be updated to support the needed security properties. This simply does not scale, as most organizations lack the developer resources to maintain the security of all legacy software that they have purchased or installed. In contrast, layering security using SDP does scale, allowing a single developer team to add additional access control in front of all legacy network services.

## IV. Related Work

A large number of academic systems have used global control points to impose security on unmodified applications. We sample a few of these here, illustrating the use of a wide variety of global control points.

Within the operating system, Wolthusen proposed layering end-to-end encryption onto email. [11]. In their system, network traffic is intercepted, email is identified, then it is encrypted and signed. Their approach requires no changes to any email clients on the system.

Within the browser, He et al. proposed ShadowCrypt, a Google Chrome extension for layering end-to-end encryption on top of web applications. [12]. ShadowCrypt mimics MessageGuard's functionality, but instead of leveraging security overlays it uses the Shadow DOM, an upcoming `HTML5` standard.

Lau et al. use a somewhat unique global control point—the accessibility framework in modern operating systems—to design Mimesis Aegis (M-Aegis), a system that layers end-to-end encryption on top of mobile applications [13]. M-Aegis functions similarly to MessageGuard but is the first system that attempts to provide ubiquitous and integrated encryption outside of the browser. While this approach would be difficult deploy across all operating systems [8], it has the potential to be far wider reaching than MessageGuard.

Bate's et al. proposed CertShim, a system that uses the operating system's dynamic library loader to layer correct TLS functionality onto applications [14]. This is accomplished by using the `LD_PRELOAD` environment variable to override functions in dynamically-loaded security libraries, such as OpenSSL. The new library ensures that TLS connections are properly authenticated, regardless of mistakes in the application's code, while also allowing alternative methods to be used that don't necessarily rely on Certificate Authorities. CertShim was later improved upon by TrustBase, which moves the global control point to the operating system, providing coverage to all applications, not just those that use certain SSL libraries. This demonstrates the importance of choosing appropriate global control points to maximize coverage.

Finally, the danger presented by bad implementations of the layered security paradigm has been illustrated by Durumeric et al. [38]. They studied a number of popular middleboxes and client-side security software that intercept TLS traffic to provide web filtering and content analysis (e.g., virus scanning). They found that nearly all of these solutions reduced the security of the TLS connection or introduced vulnerabilities. However, we note that the Blue Coat Proxy was correctly implemented, and was able to increase security as compared to unmodified connections. This demonstrates that it is possible for security experts with experience and resources are able to build security services that are robust to errors. Moreover, fixing this handful of systems with bad implementations—which could then increase the strength of TLS and Web security across all connections—is still far more manageable than fixing all applications which improperly implement TLS.

## V. Summary and Future Work

Requiring all applications to independently and correctly implement security is not working. Instead, we advocate for the layered security paradigm where security is layered on top of existing software at global control points. This paradigm greatly increases the correctness of security protecting applications and simultaneously ensures that this correct functionality applies to all applications, including those that were not designed with security in mind. Additionally, this paradigm ensures that new security features and threat mitigations can be more rapidly deployed across all applications.

In this paper, we described MessageGuard and software-defined perimeter, two examples of layering on security at a global location. MessageGuard added end-to-end encryption to existing web applications, and a software-defined perimeter adds additional access control to legacy network services. In both cases, layering security at a global location was able to support the vast majority of existing applications. Moreover, the usability studies of MessageGuard demonstrate that even user-facing security features can effectively be layered from a global location.

While the layered security paradigm has risks and may not work in all situations, in many cases, it can still substantially increase security. Importantly, it scales in a way that the current secure software development paradigm does not. As such, we believe that the community would be well served to explore additional applications for the layered security paradigm. Below are several ideas for future work.

**Exploring additional global control points.** We have discussed a variety of potential global control points, and there are certainly others possible, such as a compiler, virtual machine, and so forth. A good first step would be understanding what kinds of security enhancements are possible at each global control point, as well as the tradeoffs of pursuing a certain enhancement at different global control points.

**Layering-aware applications.** The design of layering-aware applications should be further explored, as these applications have the potential to be significantly more secure and usable that non-layering-aware applications.

**Password monitoring.** Password systems can be strengthened by monitoring systems that alert users to any misuse of

their password, similar to what Chrome's browser extension does. Alternatively, by mixing this approach with layering-aware applications, it would be possible to centralize password entry in the operating system [18]. This approach has many potential benefits and is an interesting area of future research.

**Network connection security.** Recent systems like CertShim [14] and TrustBase [10] are designed to provide a layer of protection against applications that fail to establish secure connections using proper certificate validation. They also provide hooks to introduce alternative certificate validation schemes without modifying user applications. There may be other ways to strengthen authentication leveraging this approach.

**Content-based encryption and signing of web content.** Afanasyev et al. [39] recently proposed a new security paradigm for the Web that replaces the connection-oriented security mechanisms in common use today (e.g., TLS) with a content-based security model where all data is signed and encrypted at rest. This approach could be realized with new functionality provided at global control points.

## ACKNOWLEDGMENTS

We thank Jeff Andersen, Ben Burgon, Luke Dickinson, Kimball Germane, Scott Heidbrink, Travis Hendershot, Nathan Kim, Tyler Monson, Mark O'Neill, Chris Robertson, Ryan Segeberg, Timothy van der Horst, Elham Vaziripour, Justin Wu, and Song Yuanzheng for their work on MessageGuard. We thank Mark O'Neill, Scott Heidbrink, Jordan Whitehead, Dan Bunker, Luke Dickinson, Travis Hendershot, and Joshua Reynolds for their work on TrustBase.

The MessageGuard research is supported by the National Science Foundation under Grant No. CNS-1528022. The TrustBase research is supported by the National Science Foundation under Grant No. CNS-1528022 and research sponsored by the Department of Homeland Security (DHS) Science and Technology Directorate, Cyber Security Division (DHS S&T/CSD) via contract number HHSP233201600046C.

## REFERENCES

[1] J. R. Stark, "There simply are not enough cyber-security specialists," September 2016, [Online; Posted 2016/09/27]. [Online]. Available: https://www.complianceweek.com/blogs/john-reed-stark/there-simply-are-not-enough-cyber-security-specialists

[2] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the usability of cryptographic apis," 2017.

[3] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating SSL certificates in non-browser software," in *ACM Conference on Computer and Communications Security*, 2012.

[4] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory love Android: An analysis of Android SSL (in) security," in *ACM Conference on Computer and Communications Security*, 2012.

[5] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations," in *IEEE Symposium on Security and Privacy (SP)*, 2014.

[6] L. Onwuzurike and E. De Cristofaro, "Danger is my middle name: experimenting with ssl vulnerabilities in android apps," in *ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2015.

[7] S. Ruoti, N. Kim, B. Burgon, T. van der Horst, and K. Seamons, "Confused johnny: When automatic encryption leads to confusion and mistakes," in *Proceedings of the Ninth Symposium on Usable Privacy and Security*, ser. SOUPS '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:12. [Online]. Available: http://doi.acm.org/10.1145/2501604.2501609

[8] S. Ruoti, J. Andersen, T. Monson, D. Zappala, and K. Seamons, "Messageguard: A browser-based platform for usable, content-based encryption research," *arXiv preprint arXiv:1510.08943*, 2015.

[9] S. Ruoti, J. Andersen, T. Hendershot, D. Zappala, and K. Seamons, "Private webmail 2.0: Simple and easy-to-use secure email," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ser. UIST '16. New York, NY, USA: ACM, 2016, pp. 461–472. [Online]. Available: http://doi.acm.org/10.1145/2984511.2984580

[10] M. O'Neill, S. Heidbrink, S. Ruoti, J. Whitehead, D. Bunker, L. Dickinson, T. Hendershot, J. Reynolds, K. Seamons, and D. Zappala, "TrustBase: An architecture to repair and strengthen certificate-based authentication," in *Proceedings of the 26th USENIX Security Symposium*, 2017.

[11] S. D. Wolthusen, "A distributed multipurpose mail guard," in *IEEE Systems, Man and Cybernetics Society Information Assurance Workshop*. IEEE, 2003, pp. 268–275.

[12] W. He, D. Akhawe, S. Jain, E. Shi, and D. Song, "ShadowCrypt: Encrypted web applications for everyone," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1028–1039.

[13] B. Lau, S. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva, "Mimesis aegis: A mimicry privacy shield–a system's approach to data privacy on public cloud," in *Proceedings of the 23rd USENIX Security Symposium*. USENIX Association, 2014, pp. 33–48.

[14] A. Bates, J. Pletcher, T. Nichols, B. Hollembaek, D. Tian, K. R. Butler, and A. Alkhelaifi, "Securing SSL certificate verification through dynamic linking," in *ACM Conference on Computer and Communications Security (CCS)*, 2014.

[15] L. Popa, A. Ghodsi, and I. Stoica, "HTTP as the narrow waist of the future internet," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 6.

[16] J. Howell, B. Parno, and J. R. Douceur, "Embassies: Radically refactoring the web." in *NSDI*, 2013, pp. 529–545.

[17] S. Tang, H. Mai, and S. T. King, "Trust and protection in the Illinois browser operating system." in *OSDI*, 2010, pp. 17–32.

[18] S. Ruoti and K. Seamons, "End-to-end passwords," in *Proceedings of the 2017 New Security Paradigms Workshop*. ACM, 2017.

[19] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS meets CDN: A case of authentication in delegated service," in *Thirty-Fifth IEEE Symposium on Security and Privacy (S&P 2014)*. San Jose, CA: IEEE Computer Society, 2014, pp. 67–82.

[20] D. Stefan, E. Z. Yang, P. Marchenko, A. Russo, D. Herman, B. Karp, and D. Mazieres, "Protecting users by confining JavaScript with COWL," in *Eleventh USENIX Symposium on Operating Systems Design and Implementation (OSDI 2014)*. Broomfield, CO: USENIX Association, 2014, pp. 131–146.

[21] I. D. Foster, J. Larson, M. Masich, A. C. Snoeren, S. Savage, and K. Levchenko, "Security by any other name: On the effectiveness of provider based email security," in *Twenty-Second ACM SIGSAC Conference on Computer and Communications Security (CCS 2015)*. Denver, CO: ACM, 2015, pp. 450–464.

[22] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman, "Neither snow nor rain nor MITM...: An empirical analysis of email delivery security," in *Fifteenth ACM Internet Measurement Conference (IMC 2015)*. Tokyo, Japan: ACM, 2015, pp. 27–39.

[23] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kaafar, "TLS in the wild: An internet-wide analysis of TLS-based protocols for electronic communication," in *Twenty-Fourth Network and Distributed System Security Symposium (NDSS 2016)*. San Diego, CA: The Internet Society, 2016.

[24] C. Robison, S. Ruoti, T. W. van der Horst, and K. E. Seamons, "Private facebook chat," in *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*. IEEE, 2012, pp. 451–460.

[25] A. Bangor, P. Kortum, and J. Miller, "An empirical evaluation of the System Usability Scale," *International Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

[26] ——, "Determining what individual SUS scores mean: Adding an adjective rating scale," *Journal of Usability Studies*, vol. 4, no. 3, pp. 114–123, 2009.

[27] S. Ruoti, J. Andersen, S. Heidbrink, M. ONeill, E. Vaziripour, J. Wu, D. Zappala, and K. Seamons, ""We're on the same page": A usability study of secure email using pairs of novice users," in *Thirty-Fourth ACM Conference on Human Factors and Computing Systems (CHI 2016)*. San Jose, CA: ACM, 2016, pp. 4298–4308.

[28] S. Ruoti, J. Andersen, D. Zappala, and K. Seamons, "Why johnny still, still can't encrypt: Evaluating the usability of a modern pgp client," *arXiv preprint arXiv:1510.08555*, 2015.

[29] G. Keizer, "Hackers spied on 300,000 Iranians using fake Google certificate," http://www.computerworld.com/article/2510951/cybercrime-hacking/hackers-spied-on-300-000-iranians-using-fake-google-certificate.html, accessed: 27 October, 2015.

[30] M. Marlinspike, "SSL and the future of authenticity," *Black Hat USA*, 2011.

[31] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in *Internet Measurement Conference*, 2013.

[32] P. Eckersley and J. Burns, "The (decentralized) SSL observatory," in *USENIX Security Symposium*, 2011.

[33] C. Soghoian and S. Stamm, "Certified lies: Detecting and defeating government interception attacks against SSL (short paper)," in *Financial Cryptography and Data Security*. Springer, 2012, pp. 250–259.

[34] P. Hoffman and J. Schlyter, "The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA, RFC 6698," https://datatracker.ietf.org/doc/rfc6698, 2012, accessed: 24 Feb, 2014.

[35] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving SSH-style host authentication with multi-path probing." in *USENIX Annual Technical Conference*, 2008, pp. 321–334.

[36] C. S. Alliance, "Software defined perimeter," https://cloudsecurityalliance.org/group/software-defined-perimeter/#_overview, accessed: June 2017.

[37] D. Conde and F. Benedetto, "Software-defined perimeters: An architectural view of sdp," http://sdn.ieee.org/newsletter/march-2017/software-defined-perimeters-an-architectural-view-of-sdp, accessed: June 2017.

[38] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The security impact of HTTPS interception," in *Network and Distributed Systems Symposium*, 2017.

[39] A. Afanasyev, J. A. Halderman, S. Ruoti, K. Seamons, Y. Yu, D. Zappala, and L. Zhang, "Content-based security for the web," in *Proceedings of the 2016 New Security Paradigms Workshop*. ACM, 2016, pp. 49–60.