Literature Review Template and Rubric

In this review, provide a careful, critical, and thorough explanation of relevant prior work on your project. It should be divided into two parts: background and previous work. The background is the basic material that you need to know to begin your research. Previous work builds on the background and is research that is directly relevant to your project. Your focus should be on the previous work. Do not simply list previous work; critically evaluate it and explain how your work is different and better.

Here is the grading rubric for the literature review. It will be graded on a 10 point scale, distributed as below.

- Use of Previous Work (*3 points*)
  - Work is cited properly — formatting, proper attribution of knowledge, etc. (*1 point*)
    - 1 point: mostly or all correct citation
    - 0 points: little or no correct citation
  - Work is critically analyzed and shown to be relevant — its contribution to the field and to the problem at hand is mentioned and qualified (*2 points*)
    - 2 points: most or all of the work is given adequate treatment
    - 1 point: some of the work is given adequate treatment, most or all of the work is partially addressed
    - 0 points: work is mentioned, little or no critical thought or framing is attached to the mentions
- Evidence of an argument built into the review (*7 points*)
  - Argument is well-structured — similar points grouped, logical progression, showcases relevant sides of the issue; the dimensions mentioned in the problem statement should be tackled here, and vice versa (*2 points*)
    - 2 points: the argument mostly flows logically and transitions between points are handled
    - 1 point: the argument is there but there is little flow or transitioning
    - 0 points: no evidence of organization of the argument
  - Work referenced supports the argument — each work cited makes a clear point about the state of knowledge in this area, the established procedure for investigating it, and so forth, and the utility of each work in characterization of the problem or solution is apparent. (*3 points*)
    - 3 points: the point is clear for all or almost all works
    - 2 points: the point is clear for most works, or a little more effort is needed on the clarity or justification
    - 1 point: some works, or quite a bit of effort is needed on clarity or justification
    - 0 points: most works are not given any reason to be mentioned
  - Argument supports the proposed work — each step in the proposed work is supported by precedence in literature or a demonstrated and noted lack of precedence in literature. (*2 points*)
    - 2 points: evidence is clear and clearly stated for most or all steps
    - 1 point: statement or justification is not clear, and/or some steps are not justified
    - 0 points: literature review seems to have little to do with justification for proposed work

Software Target-Focused Flow Analysis

Anonymous

STATIC ANALYSIS

FlowDroid [2] is a sophisticated context-,flow-, field-, object-sensitive and lifecycle-aware static taint analysis tool for Android applications. It does not model the inter-component communication of Android applications and hence it cannot exam the implicit information flows. Epicc [8] statically resolves the destinations of each inter-component communication instance by reducing the problem into a traditional program analysis problem. IccTA [7] and DidFail [6] combine FlowDroid with Epicc and seek to recognize sensitive inter-component and inter-application information flows. DroidSafe [5] focus on providing more accurate modeling of Android system, which identifies some data flows missed by FlowDroid due to inaccuracy modeling of life-cycle and callback events.

Above approaches inherit the general limitations of static analysis i.e. they generate false alarms. These tools are oblivious to reflective calls, native code and multi-threading and generate false alarms. Also, in preliminary experiments running FlowDroid on real apps collected from the app markets, most of them terminate in unexpected ways: either due to timeout or out-of-memory exception.

DYNAMIC ANALYSIS

Dynamic analysis is embedded into the mobile operating systems and monitors the applications at runtime. TaintDroid [4] is a sophisticated dynamic taint-tracking tool. BayesDroid [9] is built upon TaintDroid to further classify information leakages through checking values between sources and sinks.

Compared to static analysis, dynamic analysis does not need to worry about the reflections inside the apps and only report the malicious behaviors during runtime, which generate much less false alarms and is more usable to the end users. However, low code coverage limits the overall detection rate of dynamic analysis.

HYBRID ANALYSIS

To overcome the disadvantages and preserve the advantages of both static analysis and dynamic analysis, AppAudit [10] first attempts to model data flow inside apps by integrating static analysis and dynamic analysis. It quickly obtains rough over-estimated relationships between sources and sinks through static analysis, and then prunes the results through dynamic analysis. The outcomes show that it achieves better performance in both precision and usability, as compared to a pure static analysis approach or a dynamic analysis approach. Inspired by AppAudit, we attempt to come up with a better hybrid approach to extract program dependencies in the apps.

We notice that AppAudit still misses a lot data flows on DroidBench [1], which is a benchmark toolkit to test the performance of taint analysis tools. The main reason behind is AppAudit purely relies on concrete value to proceed the analysis. Unknown value, however, may be generated due to the insufficient run-time information. The existence of unknown value in the conditional program statements will lead to incomplete searching paths and inaccurate data flow results. The problem can be alleviated with the help of symbolic execution [3]. Symbolic execution leverages symbolic representation and first-order logic to represent an unknown variable. We will integrate the ideas from symbolic execution into the technique proposed by AppAudit to achieve higher accuracy.

Also, the call graph algorithm of AppAudit does not consider the potential sensitive behaviors inside Android GUI callbacks. We can create a more appropriate model of the mobile operating system to locate the stealthy behaviors.

Since the core engine of AppAudit is not open-source, we also need to implement our own virtual machine and program analysis module.

REFERENCES

[1] Droidbench. `http://blogs.uni-paderborn.de/sse/tools/droidbench/`. Accessed: 2016-02-26.

[2] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Proc. of PLDI*. ACM, 2014.

[3] C. Cadar and K. Sen. Symbolic execution for software testing: three decades later. *Communications of the ACM*, 56(2):82–90, 2013.

[4] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N.

Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[5] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard. Information flow analysis of android applications in droidsafe. In *NDSS*, 2015.

[6] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer. Android taint flow analysis for app sets. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, pages 1–6. ACM, 2014.

[7] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 280–291. IEEE Press, 2015.

[8] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. Le Traon. Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. *Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis*, 2013.

[9] O. Tripp and J. Rubin. A bayesian approach to privacy enforcement in smartphones. In *Proc. of USENIX Security*, 2014.

[10] M. Xia, L. Gong, Y. Lyu, Z. Qi, and X. Liu. Effective real-time android application auditing. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 899–914. IEEE, 2015.

MEG Literature Review

Anonymous

1 Introduction

Few people realize today that the vast majority of emails transmitted today are sent completely unencrypted. This allows anyone who has access to the routers these communications are being sent over to read and even modify the contents of these communications. Encryption is a technology that is used liberally on the internet today. HTTPS ensures that bank communications are kept secure. SSH supplanted telnet because other actors cannot view and modify what actions that administrators perform on remote machines. So why is email an outlier? The problem does not lie in the infeasibility of encrypting email but rather the practicality of it. Existing email encryption schemes are clunky and can only be performed by committed users. This leaves the vast majority of non-technical and less security conscious users completely naked to their emails being spied. The Mobile Encryption Gateway (MEG) aims to solve this. MEG aims to take the difficulty of encrypting email and perform it all in the background while allowing the user to have as much ease as possible in performing secure communications. This is combined with an aspect of security and data privacy that MEG provides where all communications remain encrypted on a users mobile device. As a result of this security and ease of use MEG offers ordinary people their best chance going forward to have a reasonable encryption solution for their emails.

2 Existing Encryption Schemes

2.1 For starters: PEM

Privacy Enhanced Mail (PEM) was the first encryption scheme to be released for email. PEM worked whereby users could publish their public keys and then these keys could be signed by the private key of a certifying authority. In PEM Certificate Authorities (CAs) worked by creating a hierarchical system of trust eventually terminating in a single trusted root authority, the Internet

Policy Registration Authority (IPRA) [11, 15]. However PEM was never widely implemented when the security concerns and potential liabilities of a single root authority became evident [2, 17]. As a result of this S/MIME was created to address the deficiencies of PEM.

2.2 S/MIME to the rescue?

After the abandonment of PEM S/MIME became the de-facto email encryption standard. The main difference between S/MIME and PEM is S/MIME drops the necessity for a single root authority and acknowledges the existence of multiple independent CAs [11]. It is for this reason that S/MIME has been able to integrate itself into the web given that the major CAs today are well accepted. The problem with S/MIME is the issue of certificate creation. Creation of X.509 certificates is a centralized process imposed by the CAs [12]. The CAs charge expense for their services, anathema to many users used to the many free services available online. In addition the process of obtaining a certificate can take days if not weeks. Certificates are specific to an email address meaning that each individual user must obtain their own personal certificate for their mail account if they do not have a system administrator able to obtain it for them. Even then, network effects must be taken into account and if the recipient of an encrypted email does not have an S/MIME certificate themselves then the email cannot be sent in encrypted form. It is for these reasons that users have found the process of obtaining S/MIME certificates onerous to the point of infeasible [12].

The best attempt yet to alleviate these usability issues with S/MIME is a solution created by Simson Garfinkel named Key Continuity Management (KCM). KCM operates by directly creating a self signed X.509 key pair for a user and then injecting that into Microsoft Outlook. Messages can then be signed and encrypted. KCM manages this entire process in the background making it seamless for an user to adopt. Because the keys are self signed trust is given to the host from which the message was received instead of a CA [12]. This allows KCM to operate both instantaneously and for free in comparison to a CA. Yet the abandonment of CAs leads to major flaws with this idea. The first problem with this scheme arises in the self signing of certificates. By Garfinkel's own admission many mail clients discourage a user from accepting an email that is self-signed [11]. In addition many mail clients do not even accept the use of self-signed certificates, viewing them as insecure [14, 16]. Garfinkel tries to argue that the use of self signed certificates would be more acceptable if the user could determine whether to trust the host the message is coming from similar to how SSH works [11]. However this does not constitute a feasible solution to the problem. Garfinkel is essentially stating that since self signed certificates will never be able to validate the identity of who is sending the communication [9] let the user determine who to trust. But asking the user to understand their risk is unreasonable. The vast majority of users do not understand the purpose of self signed certificates [7]. When study participants were prompted with security decisions in a study on HTTPS on whether or not to accept dubious certificates

users frequently chose to accept them [5]. In fact, according to Downs a likely way to spoof identity would just be to impersonate a company the user does business with [7]. Given how easy users are to fool we see why KCM would face suspicion from regular email providers like Google and Microsoft. Furthermore KCM adoption would place users at greater security risk given they will trade in provider built-in phishing protection and spam filtering for encrypted mail.

To conclude our discussion of S/MIME we note that it is the current leading standard for email encryption. But this does not mean that it constitutes the best scalable solution. S/MIME has major usability and cost concerns associated with using CAs as a source of validating contact authenticity. Attempts to remedy these concerns through KCM enable widespread encryption but take a major step backwards on validating contact identity and user security. If one wants to use S/MIME then using the CAs is the only way to correctly perform encryption. Yet even if someone was able to automate usability concerns away cost issues would still persist. Thus S/MIME cannot truly be held as a standard for widespread email encryption going forward.

2.3 PGP

The main alternative to S/MIME is PGP. In a perfect world users would just be able to perform the service of encrypting and validating all mail for themselves solely using PGP. In contrast to S/MIME X.509 certificates PGP keys are decentralized. Instead of a single certificate authority to vouch for the identity of a person PGP operates over a web of trust between other people. The projects creator Phil Zimmermann claimed that eventually PGP "will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys." [20] In practical terms the web of trust works through key signing. The greater amounts of times a key has been signed, or certified, as trustworthy by other users the more trustworthy that key is [10]. This means that with proper use of the web of trust PGP is able to validate the authenticity of the person one is communicating with [9]. This is a major improvement over self-signed X.509 certificates.

In further contrast with S/MIME even an inexperienced user can generate a PGP key in a matter of minutes on their own personal computer [19, 20]. PGP when properly used guarantees that communications will be completely unreadable to parties not intended to receive a message [20]. With this known academics decided to perform user studies on how well people were able to encrypt their emails when using PGP. In disappointment to security advocates a seminal study named "Why Johnny can't Encrypt" decisively found the majority of users are incapable at encrypting messages using PGP even given ample amounts of time to try to perform the task [19]. Even using the best GUI interface for PGP at the time, PGP 5.0, the few who were able to encrypt their communications often exposed their private key in plain text [19]. A follow-up study conducted in 2006 on a more modern interface still found users could not accomplish the task of encrypting and sending an email [18]. Results for another study in 2016 will doubtfully be much different. And while dedicated

security and IT personnel will likely have little trouble in encrypting communications regular nontechnical persons cannot be expected to perform native PGP encryption even using a GUI. In summary, while PGP offers the benefits of decentralized encryption based on a web of trust it cannot natively offer a scalable solution for email encryption.

2.4 Alternatives

As a result of difficulties with PGP and S/MIME alternative schemes for email encryption have surfaced. A lightweight key distribution system [3] is one such system devised. A proxy gateway that has knowledge of the user's private key is another system that can be adapted for email encryption [4]. The main problem with these systems is that they will be implemented by the mail provider. We now know post-Snowden that mail providers sometimes collaborate with the government to release private user data. If provider and government are not partners we saw in the case of Lavabit it is trivial for the government to obtain a persons data even if the government does not have a warrant for that specific user. A service merely has to be of suspicion and that will cause the release of private information [1].

There are paid services that take the usability headaches of S/MIME and PGP and automate that for individual users. Services like ciphermail have built in CAs [6] so they can issue X.509 certificates instantly. Hushmail uses PGP to perform encryption [13]. However paid services are still not necessarily secure even if they provide encryption for mail in transit. For example Hushmail only receives a score of 1 out of 7 from the Electronic Frontier Foundation for its services [8]. This is due to the fact the service does not offer end to end encryption, users cannot verify the identity of contacts, and past messages aren't secure if encryption keys are stolen among other things. Most importantly these services all still suffer from the chicken and the egg problem of the network effect that afflicts both S/MIME and PGP. If users are using a paid encrypted service then their emails will undoubtedly be encrypted. But if people are not using a paid service then there is small chance they will be enticed to spend their money on one just to receive encrypted email. Indeed why bother spending money on encryption when Gmail is free.

Services like Hushmail and ciphermail provide the easiest alternative for users who want their emails encrypted in transit. Email services like Gmail could conceivably add additional encryption techniques onto their service like a proxy to allow encryption. Yet, these solutions are not perfect. Hushmail has a less than stellar privacy rating. The fact that all paid clients do not have open source code means we cannot validate the security of the services provided. And using a proxy with a shared key is not guaranteed as a means of protecting against government or criminal intrusion. A viable solution for email encryption needs to ensure that the users private key stays private, that a service maintains complete end to end encryption, and that the code be open source or at the least be continuously validated in security audits.

3 MEG; and how it fixes things

MEG aims to address many of the problems that stem from using PGP, S/MIME, and mail provider based encryption schemes. First and foremost MEG aims to use PGP because it is free, decentralized, and can validate contact identity using web of trust. MEG however aims to alleviate the usability headaches surrounding PGP by automating the entire process of key generation, signing keys, encryption, and decryption of messages for the user. This way the user gets to enjoy encryption for their emails while not having to worry about the low level details of how it works.

MEG will ensure user data stays private. End to end encryption will be built into MEG on any part where it is necessary. A users private key will stay on their phone. This way no one except the owner of the phone will have the ability to decrypt messages. The actual encryption of emails will be provided as a service by a mobile phone with the MEG Android app on the mobile device serving as the email gateway. MEG will then relay the emails over a secure channel to the MEG client which can forward the email to its intended destination.

Web of trust is probably the key concept of PGP which separates it from X.509. It will also be the hardest thing to get right in designing MEG. So we are going to try to make the users life as painless as possible when navigating the web of trust. Emails will be color coded depending on their trust level. Green coding will be attached to email that we explicitly trust from a user. Yellow will be coded for emails we cannot necessarily validate but rather ones that are trusted through the web of trust. Red coding will be attached to emails the are either untrusted or are not encrypted. This way a user can have a more graphical representation on who to trust and who not to. In fact this scheme has already been used by KCM and actually worked fairly successfully in usability studies [12].

MEG has the benefit of being able to learn from its predecessors. We have the ability to understand where previous schemes such as KCM failed. And we also have the benefit of being able to use the ubiquity of smartphones to serve as email gateways. Email encryption can be a reality for anyone who wants it. It is just a matter of engineering the correct solution that people can use. We believe that MEG is that solution.

References

[1] Lavabit. https://en.wikipedia.org/wiki/Lavabit. Accessed: 2016-02-12.

[2] Privacy-enhanced electronic mail. https://en.wikipedia.org/wiki/Privacy-enhanced_Electronic_Mail. Accessed: 2016-02-12.

[3] Ben Adida, Susan Hohenberger, and Ronald L Rivest. Lightweight encryption for email. In *SRUTI*, 2005.

[4] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology—EUROCRYPT'98*, pages 127–144. Springer, 1998.

[5] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security and Privacy*, 7(1):78–81, 2009.

[6] ciphermail. Email encryption gateway. https://www.ciphermail.com/gateway.html. Accessed: 2016-02-17.

[7] Julie S Downs, Mandy B Holbrook, and Lorrie Faith Cranor. Decision strategies and susceptibility to phishing. In *Proceedings of the second symposium on Usable privacy and security*, pages 79–90. ACM, 2006.

[8] Electronic Frontier Foundation. Secure messaging scorecard. https://www.eff.org/secure-messaging-scorecard. Accessed: 2016-02-17.

[9] Steven M Furnell, Nathan Clarke, Cristian Thiago Moecke, and Melanie Volkamer. Usable secure email communications: criteria and evaluation of existing approaches. *Information Management & Computer Security*, 21(1):41–52, 2013.

[10] Simson Garfinkel. *PGP: pretty good privacy.* " O'Reilly Media, Inc.", 1995.

[11] Simson L Garfinkel, David Margrave, Jeffrey I Schiller, Erik Nordlander, and Robert C Miller. How to make secure email easier to use. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 701–710. ACM, 2005.

[12] Simson L Garfinkel and Robert C Miller. Johnny 2: a user test of key continuity management with s/mime and outlook express. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 13–24. ACM, 2005.

[13] Hushmail. How hushmail works. https://www.hushmail.com/about/technology/how-it-works/. Accessed: 2016-02-17.

[14] Lee Hutchinston. Taking e-mail back, part 2: Arming your server with postfix and dovecot. http://arstechnica.com/information-technology/2014/03/taking-e-mail-back-part-2-arming-your-server-with-postfix-dovecot/. Accessed: 2016-02-17.

[15] S. Kent. Privacy enhancement for internet electronic mail: Part ii: Certificate-based key management. https://www.ietf.org/rfc/rfc1422.txt. Accessed: 2016-02-17.

[16] Howard Lightstone. How can i force use of self-signed ssl certificate. https://productforums.google.com/forum/!topic/gmail/6gODk9n65ZU. Accessed: 2016-02-17.

[17] Bryan D Payne and W Keith Edwards. A brief introduction to usable security. *Internet Computing, IEEE*, 12(3):13–21, 2008.

[18] Steve Sheng, Levi Broderick, Colleen Alison Koranda, and Jeremy J Hyland. Why johnny still can't encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security*, pages 3–4, 2006.

[19] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Usenix Security*, volume 1999, 1999.

[20] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.