Meet the new threat model, same as the old threat model

Eric Rescorla
Mozilla
ekr@rtfm.com

January 20, 2014

> The Internet environment has a fairly well understood threat model. In general, we assume that the end-systems engaging in a protocol exchange have not themselves been compromised. Protecting against an attack when one of the end-systems has been compromised is extraordinarily difficult. It is, however, possible to design protocols which minimize the extent of the damage done under these circumstances.
>
> By contrast, we assume that the attacker has nearly complete control of the communications channel over which the end-systems communicate. This means that the attacker can read any PDU (Protocol Data Unit) on the network and undetectably remove, change, or inject forged packets onto the wire. This includes being able to generate packets that appear to be from a trusted machine. Thus, even if the end-system with which you wish to communicate is itself secure, the Internet environment provides no assurance that packets which claim to be from that system in fact are. — *RFC 3552 [RK03]*

> It is a terrible thing for a man to find out suddenly that all his life he has been speaking nothing but the truth.— *The Importance of Being Earnest*

1 Introduction

As suggested by the above quote, historical Internet and IETF security practice has focused on two major avenues:

- Hardening end-user software so that it is not subject to remote compromise.
- Designing, implementing, and deploying secure communications protocols (the primary area where the IETF works).

Neither of these efforts has been spectacularly successful. Even software developed by organizations with very aggressive security programs regularly has vulnerabilities discovered and while the Internet security community has developed an array of reasonably secure protocols—though even these have been subject to a variety of more-or-less successful attacks—the vast majority of Internet communications remain in plaintext.

Despite increasing levels of effort by software developers, protocol designers, operators, etc. this was the situation in mid-2013 when Edward Snowden started releasing a series of documents which described alleged NSA surveillance activities on Internet traffic. Barnes et al. [BSJH14] describe a number of the allegations, which include:

- Wide-scale capture of Internet traffic.
- Remotely exploiting software on users and operators computers to install spyware.
- Intercepting computing devices in shipment to users to install hardware monitoring devices.
- Promoting the standardization of compromised cryptographic algorithms.

To a great extent, these allegations (if true) reinforce rather than contradict the threat model documented in RFC 3552 [RK03]: we must be concerned with attackers with complete control of the network who, at least at times, also have control of our endpoints. The new information here (if any) is that the threat model that security people have been urging developers to adopt is real, not hypothetical. Unfortunately, while this information potentially shifts the balance of cost and benefit in terms of trying to make things more secure, it doesn't address any of the fundamental problems (strong identity, installed base, the difficulty of writing secure code, C/C++ (but I repeat myself), etc.) that have made it so difficult to secure the Internet in the past.

2 What's reasonable to defend against?

> **William Turner**: You didn't beat me. You ignored the rules of engagement! In a fair fight, I'd kill you!
> **Jack Sparrow**: Well, that's not much incentive for me to fight fair then, is it?
> — *Pirates of the Caribbean: The Curse of the Black Pearl*

The major new element in the threat model is the claim that the NSA has a systematic program of mounting direct physical attacks on target's computers. This form of attack is conceptually distinct from remote attack because it's not really even potentially preventable by better software or hardware engineering, short of fabricating totally new hardware from the ground up, including a completely assured chain of custody from the point of initial manufacture to the point of use. This might be plausible for very restricted use cases, but not for equipment for the average user.[1] Of course this form of attack has always been known to be possible, but it's different to have public reports that it's actually happening.

Fortunately, this sort of attack—at least at the retail level—seems like it's not that scalable an attack model, as it requires contact with the subject's computer. Of course, this limitation does not apply if the attacker has direct control of the design and/or manufacturing process (with or without the cooperation of the manufacturer) as contemplated by King et al [KTC+08]. In any case, as we do not have a plausible defense against this form of attack our best option is to design systems under the assumption that the people we are trying to protect are not being so targeted Rather, we should be focusing our efforts on defending against attacks that can be mounted remotely and scalably.

3 What is to be done?

> The only rules that really matter are these: what a man can do and what a man can't do.
> — *Captain Jack Sparrow*

At some level, "what is to be done" is "more of the same"; this is especially true of the IETF, which principally deals with protocols and not end-systems and has already expended very considerable effort on designing security for those protocols. However, this does not mean that nothing could be done better. This section outlines a few short-to-medium term steps for the Internet security community and the IETF to consider.

3.1 Prioritize Deployment

Any fair examination of the Internet protocol suite inevitably comes to the conclusion that the availability of "secure" protocols far outstrips their actual deployment. We have "secure" versions of at least the following functions:

- All IP traffic (IPsec)

[1] It is customary at this point to cite Thompson's original observation of this problem in the context of software [Tho84], but the problem of hardware is far worse, especially as so much of the software stack has in fact been rebuilt from the ground up largely in the open. This is not of course say that our software development process is anywhere near close to sufficient to detect malicious code inserted by developers, but it is at least in principle largely detectable.

- Web browsing (HTTPS)
- Remote login (SSH)
- E-mail content (OpenPGP, S/MIME), delivery (SMTP over TLS) and retrival IMAP and POP over TLS)
- Voice and video over IP (SRTP and the pile of key management protocols that go with it.)

In the vast majority of these cases, the "insecure" (i.e., unencrypted and unauthenticated) version of the protocol is still in wide use and at least in the cases of IP, Web, standards-based voice and video, and e-mail content, is the dominant version. Secure protocols don't help if they aren't used and the clear lesson here is that we need to do something more than just specify secure protocols, we need to do something to make them sufficiently compelling that people will use them in preference to insecure protocols. The examples above suggest a number of lessons for how to improve deployment.

> *Make it easy and automatic.* The available evidence (e.g., [WT99]) is that people are only willing to use security mechanisms when they are easy. Thus, for instance, SSH is more-or-less of a drop-in replacement for rsh and is secure out of the box. Similarly, HTTPS is invisible for the client (though work for the server operator). By contrast all of the available secure e-mail systems require quite a lot of user engagement and have minimal usage. Indeed, probably the most popular "secure" e-mail system is HTTPS to web mail for exactly this reason.
>
> *Only make a secure version.* Painful experience suggests that if users have the choice between security and convenience, they will choose the latter. Only offering a secure version of something (as has been done with WebRTC) is one way to get them to make the other choice, provided that the application is sufficiently compelling.
>
> *Do something new or better.* Experience with IPsec suggests that getting people to convert from an insecure version of something to a secure version is very difficult unless there is some user benefit. However, if you can offer them a new capability then you can drive adoption. Examples here include port forwarding with SSH, VPN access, or voice and video (WebRTC).

The common theme in all of these points is reducing the barrier to entry for deploying secure systems. While the recent revelations about surveillance may to some extent increase people's motivation to use secure systems, it is too much to expect that they will do so unless it is either very convenient or motivated by some other need.

It's important to recognize that once a protocol is deployed, it is often possible to upgrade it to better versions without significantly impacting users, merely via the usual software upgrade process. In many cases, it is the initial deployment that is the problem. Additionally, even imperfect security protocols can increase the cost to attackers, thus lowering the overall level of attack, even if a dedicated attacker can still succeed against some targets.

3.2 Do something about Identity

> Look, let me explain something to you. I'm not Mr. Lebowski. You're Mr. Lebowski. I'm the Dude. So that's what you call me. That, or His Dudeness Duder or El Duderino, if, you know, you're not into the whole brevity thing. — *The Dude*

One of the largest barriers to deploying a secure protocol is having the identity infrastructure to allow you to identify who you are talking to. TLS (or at least HTTPS) confronts this problem head-on by requiring that servers have certificates that match their nominal domain name. This has been quite successful and resulted in very widespread deployment of TLS, but is also widely viewed as a major barrier to deploying a new HTTPS site. Moreover, there have been a number of highly-publicized cases of certificate misissuance, suggesting that the CA system is both too difficult for ordinary people to use and too easy for attackers too exploit.

SSH more or less sidesteps the problem by a combination of Trust on First Use (TOFU) with key continuity (for server authentication) and manual bootstrap by users (for client authentication). This has been extremely successful in terms of deployment—it is harder to evaluate its resistance to attack—but it's far from clear how to extend it to other systems. For instance, attempts to use a similar key continuity mechanism with HTTPS [Res00] have seen only very limited usage.

Nearly every other attempt to deploy wide-scale cryptographic authentication (e.g., end-user certificates for S/MIME, IPsec certificates, DNSSEC) has run into extremely difficult adoption problems.

Probably the most promising recent trend is the development of a variety of third-party identity protocols (e.g., Facebook Connect, OpenID Connect, OAuth, BrowserID) which are oriented towards the Web but rooted in the HTTPS CA infrastructure. The idea here is that users already have account relationships with providers which could be used for authentication in general if extended beyond the provider site. Thus, we don't have the problem of barrier to user entry. This sort of system is is one of the dominant forms of Web Authentication and efforts are actively underway to transition it to a generic identity system for protocols such as WebRTC [Res13].

3.3 Converge on Trustable Cryptographic Algorithms

The design and evalution of cryptographic algorithms is essential for the design of secure protocols, but is a distinct area of expertise. For both regulatory and organizational reasons, historical Internet communications security practice has depended heavily on NIST to standardize and promulgate these algorithms. Reports that the NIST-standardized DUAL_EC_DBRG was deliberately designed with built-in weaknesses call into question the wisdom of this practice. Unfortunately, however, while there are a number of candidate replacement algorithms for the NIST algorithms, there is no good existing mechanism for selecting a small recommended set from amongst these candidates.

The most immediate need is for a new set of elliptic curves; most ECC implementations currently use the NIST-standardized curves, but a number of concerns have been raised about the way in which they were generated and it seems likely that they at least need reevaluation and perhaps replacement. In the longer term, we will need to develop a way to collectively analyze and select algorithms that does not depend on trusting any particular organization or national standards body.

3.4 Improve Detection

Defense in depth is a basic principle of security and we know from experience that none of our security mechanisms is likely to be perfect. We are badly in need of mechanisms to detect when they fail and warn us that something is wrong. One of the better security developments in the past few years has been the use of key "pinning" by Chrome to detect bogus certificate issuance; this mechanism was the first discovery of the Diginotar breach[2]. Unfortunately, attempts to scale pinning and similar techniques beyond a few high value sites have to date not really been that successful, perhaps because they require active intervention by site operators.

There are a number of pieces of work in progress which may be useful on the detection front: Certificate Transparency[3], to create public records of every certificate issued, thus detecting bogus issuance. A number of projects have catalogued the certificates in use on the Internet (e.g., [DKBH13]) and it may somehow be possible to use this information to detect misbehavior. One challenge with any approach that attempts to allow end-user software to detect bogus certificates is reporting the problem without compromising the user's privacy. Consider the case where the user is visiting a porn site which for some reason has a suspect key. Under what conditions is it acceptable to report this back to the browser operator? Is user consent required? Obviously, from a detection perspective it is desirable to have a low threshold for suspicion, but that interacts poorly with either constant reporting or obtaining user consent for each report.

[2] http://googleonlinesecurity.blogspot.com/2011/08/update-on-attempted-man-in-middle.html

[3] `http://www.certificate-transparency.org/`

4 A Note on Metadata vs. Content

> Your disguise is as convincing as a giraffe wearing dark sunglasses trying to get into a polar bear's only golf club. — *Blackadder*

The astute reader will have noticed that this document makes no real recommendations about protecting metadata and preventing linkage. While these are clearly very important topics, the hard truth is that providing technical mechanisms to protect them is a major unsolved problem. For instance, is well-known that preventing fingerprinting of modern browsers is extremely difficult (see for instance Mowery and Shacham [MS12]) and while it is possible to reduce fingerprinting surface at the cost of functionality, we should not expect most users to actually make this choice.

Even when users are willing to make significant compromises in terms of performance and functionality in order to avoid tracking, we know that existing mechanisms such as Tor provide only imperfect protection [JWJ+13] even against non-nation-state adversaries. This is not to say that these tools are useless, and in particular they may serve to increase attack cost (e.g., by requiring more sensors or active intervention). However, we are not yet at the point where it is practical to make recommendations for ordinary users that will provide them with strong protection from tracking.

References

[BSJH14] Richard Barnes, Bruce Schneier, Cullen Jennings, and Ted Hardie. Pervasive Attack: A Threat Model and Problem Statement. draft-barnes-pervasive-problem-00, January 2014.

[DKBH13] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the https certificate ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 291–304, New York, NY, USA, 2013. ACM.

[JWJ+13] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS '13)*. ACM, 2013.

[KTC+08] Samuel T. King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou. Designing and implementing malicious hardware, 2008.

[MS12] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In Matt Fredrikson, editor, *Proceedings of W2SP 2012*. IEEE Computer Society, May 2012.

[Res00] E. Rescorla. HTTP Over TLS. RFC 2818, Internet Engineering Task Force, May 2000.

[Res13] Eric Rescorla. WebRTC Security Architecture. draft-ietf-rtcweb-security-arch-07, July 2013.

[RK03] E. Rescorla and B. Korver. Guidelines for Writing RFC Text on Security Considerations. RFC 3552 (Best Current Practice), July 2003.

[Tho84] Ken Thompson. Reflections on trusting trust. *Communications of the ACM*, 27:761–763, 1984.

[WT99] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *In Proceedings of the 8th USENIX Security Symposium*, 1999.