Security Hazards when Law is Code

by

Eric Wustrow

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2016

Doctoral Committee:

Associate Professor J. Alex Halderman, Chair
Research Professor Peter Honeyman
Associate Professor Z. Morley Mao
Professor Paul Resnick

ACKNOWLEDGEMENTS

I would not have been able to complete this dissertation or the work contained herein without the immense support I have been lucky enough to receive from my friends, colleagues, and family.

I want to express my gratitude to my advisor J. Alex Halderman for his unending inspiration, patient guidence, and comradery throughout my graduate studies. Alex has been an outstanding mentor and friend, and I have learned a great deal from him.

I would like to thank the people closest to me over the years: Mary Lin, Jenn Roloff, Jess Ouyang, Meghan Clark, and Courtney Poles for their helping me through the most difficult parts, as well as Dana Wilson for giving me the best advice I didn't know I needed.

I also thank my many labmates for keeping me company: Scott Wolchok, James Kasten, Zakir Durumeric, David Adrian, Drew Springall, Travis Finkenauer, Ben Burgess (thanks for not social engineering me out of anything important), Colleen Swanson, Allison McDonald, Ariana Mirian, Benjamin VanderSloot, and Matt Bernhard.

I'd like to thank David Devecsery for distracting me with rock climbing, and for the many research discussions we have had.

I also thank my thesis committee members Peter Honeyman, Z. Morley Mao, and Paul Resnick for their detailed feedback on my dissertation, as well as Blake Reid for his discussions on computable policy and law. Additionally, I thank Michael Bailey, Nadia Heninger, and Brian Noble for their advice and wisdom.

I am grateful to the National Science Foundation and the New America Foundation for funding me through my graduate work, as well as my co-workers and friends from Square for the experience and perspective they provided.

I also thank Lab11, especially Prabal Dutta, Ben Kempke, Pat Pannuto, Brad Campbell, Will Huang, and Branden Ghena for their seemingly infinite supply of hardware at low prices and few questions asked.

Finally, I would like to thank my family for their support, especially my mother for her unwavering care and my father for his persistant encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

As software continues to eat the world, there is an increasing pressure to automate every aspect of society, from self-driving cars, to algorithmic trading on the stock market. As this pressure manifests into software implementations of everything, there are security concerns to be addressed across many areas. But are there some domains and fields that are distinctly susceptible to attacks, making them difficult to secure?

My dissertation argues that *one domain in particular—public policy and law—is inherently difficult to automate securely using computers*. This is in large part because law and policy are written in a manner that expects them to be *flexibly interpreted* to be fair or just. Traditionally, this interpreting is done by judges and regulators who are capable of understanding the intent of the laws they are enforcing. However, when these laws are instead written in code, and interpreted by a machine, this capability to understand goes away. Because they blindly follow written rules, computers can be tricked to perform actions counter to their intended behavior.

This dissertation covers three case studies of law and policy being implemented in code and security vulnerabilities that they introduce in practice. The first study analyzes the security of a previously deployed Internet voting system, showing how attackers could change the outcome of elections carried out online. The second study looks at airport security, investigating how full-body scanners can be defeated in practice, allowing attackers to conceal contraband such as weapons or high explosives past airport checkpoints. Finally, this dissertation also studies how an Internet censorship system such as China's Great Firewall can be circumvented by techniques that exploit the methods employed by the censors themselves.

To address these concerns of securing software implementations of law, a hybrid human-computer approach can be used. In addition, systems should be designed to allow for attacks or mistakes to be retroactively undone or inspected by human auditors. By combining the strengths of computers (speed and cost) and humans (ability to interpret and understand), systems can be made more secure and more efficient than a method employing either alone.

CHAPTER 1

Introduction

In 1993, a man named John Angus Smith was arrested for attempting to purchase cocaine from an undercover officer. Instead of using cash, however, Smith attempted to trade an unloaded MAC-10 firearm for the drugs. In the following hearings, it was argued that this enacted additional punishments under U.S.C. § 924(c)(1) for when a defendant "during and in relation to ... [a] drug trafficking crime[,] uses . . . a firearm." In his appeal, Smith argued that the firearm was only a medium of exchange—not used as a weapon—and thus did not constitute "use" in the traditional sense.

This case was ultimately decided by a 1993 Supreme Court ruling [3]. Did the word "use" apply to this non-traditional employment of a gun in a drug trafficking crime? In a 6-3 decision, the court decided that indeed it did, and thus the additional sentence was applied. The majority opinion was that if Congress had wished for a more nuanced definition of the word "use", they could easily have done so in the statute. However, because they did not, it is acceptable to use a more general definition that includes "use" of a firearm as a medium of exchange. After all, the majority opinion argues, doesn't it constitute "use" of a firearm if the defendant pistol-whips a victim, but does not fire or brandish it? Surely "use" should not be limited to mean "use in its designed purpose."

The dissenting opinion argued differently: that "use" in this case should be interpreted to its ordinary meaning within context, and only apply if the firearm is used as a weapon. "To use an instrumentality ordinarily means to use it for its intended purpose. When someone asks, 'Do you use a cane?', he is not inquiring whether you have your grandfather's silver-handled walking stick on display in the hall; he wants to know whether you walk with a cane," wrote Justice Scalia in his dissent. "The Court" he argued, "does not appear to grasp the distinction between how a word *can be* used and how it *ordinarily is* used [3]."

Ultimately, this decision came down to interpreting the meaning of a single word: "use". This is not a particularly difficult word, and most English speakers would say they have a fairly good understanding of what the word means. And yet it can clearly have complicated and varied interpretations even amongst Supreme Court Justices. It is unlikely that Congress originally considered that a future defendant might trade a gun for drugs when they wrote U.S. Code § 924. Rather, it is an example of a corner case: an unexpected, uncommon application that exercises this particular code or wording in an unintended way.

This kind of mistake or oversight is easy to make when writing rules, code, or law. We do not expect lawmakers to be infinitely clever when they draft legal code, nor do we expect them to have clairvoyance over all of the future cases to which the code will apply. Instead, our legal system is configured to allow judges, jurors, and justices to *interpret* the law as written and apply it to unique and unexpected scenarios in a (literally) case-by-case basis. This provides a flexibility that is useful for resolving typos, oversights, or ambiguity of the applicable laws. Without this, would-be criminals might be able to escape justice by finding and exploiting minutiae or loopholes in laws that were not previously considered by lawmakers, or otherwise-innocent citizens could be charged on technicalities.

To avoid this, American courts often use a legal theory known as the absurdity doctrine [99] (or the Golden Rule in English courts), which emphasizes reasonable and common sense interpretation over literal execution of the law.

One such application of the absurdity doctrine is in the 1868 Supreme Court case of *United States v. Kirby*, where a mail carrier was arrested by a sheriff on a bench warrant for murder [1]. Since this arrest delayed the delivery of mail (a Federal offense), the arresting sheriff was then charged with "knowingly and willfully obstructing or retarding the passage of the mail." The Supreme Court overturned this charge, stating that it would be unreasonable and absurd to interpret this statute in this context.

"The reason of the law in such cases should prevail over its letter," Justice Field wrote in the majority opinion. "The common sense of man approves the judgment mentioned by Puffendorf that the Bolognian law which enacted, 'that whoever drew blood in the streets should be punished with the utmost severity' did not extend to the surgeon who opened the vein of a person that fell down in the street in a fit. The same common sense accepts the ruling, cited by Plowden that the statute of 1st Edward II, which enacts that a prisoner who breaks prison shall be guilty of felony does not extend to a prisoner who breaks out when the prison is on fire—'for he is not to be hanged because he would not stay to be burnt.'" [1]

This type of flexibility is not without its downsides. Indeed, the non-unanimous decision of *Smith v. United States* suggests that even seemingly simple ambiguities can sometimes escape consensus. In this case, the court held that trading a gun for drugs meant that the trader had "used" a firearm. But this wasn't the last time the meaning of that particular word came into question in the Supreme Court. In 2007, the opposite question was asked: did trading drugs for a gun also constitute "use" of the firearm in the same way?

In the unanimous decision of *Watson v. United States*, it did not [4]. The court held that contrary to *Smith v. United States*, *receiving* a gun did not count as using it, but *giving* it did. “A boy who trades an apple to get a granola bar is sensibly said to use the apple, but one would never guess which way this commerce actually flowed from hearing that the boy used the granola,” wrote Justice Souter in the majority decision [4].

Although the decision was unanimous, not all justices agreed with the reasoning behind it. Justice Ginsburg wrote a concurrence that made similar arguments to the dissent in *Smith v. United States*—that the word “use” should mean use as a weapon, not use as an economic instrument—and even went as far to say that she would overrule *Smith v. United States* and “thereby render our precedent both coherent and consistent with normal usage” [4].

These cases illustrate the difficulty in enforcing even clearly written laws. Both *Smith v. United States* and *Watson v. United States* asked subtly different versions of the same question concerning the definition of the word “use” and ended up with dramatically different results. Law is intended to be flexibly interpreted. If it were not interpreted flexibly, typos and loopholes could be exploited to bring about absurd or unjust outcomes. And because law is often meant to be a final decision or ruling, the abuse of such a system may have permanent or long-lasting consequences on individuals and society alike. It is important that the institution remain as robust, fair, and just as possible.

Law and policies are written as a kind of human-language (e.g. English) code, which is interpreted by human judges and public servants. Humans are fairly forgiving in this task, willing to automatically correct typos, recognize (if not resolve) ambiguity, and understand intent behind the code, rather than blindly follow it.

But what happens if these laws are instead written for and interpreted by computers? What if we implement laws in software code, and have computers enforce our public policies? This type of code is much more rigid, and even the most complex artificial intelligence systems do not have the kind of deep understanding and insight that our laws demand. What can go wrong when our laws and public policies are implemented in hardware and software, and blindly executed by computers?

In this thesis, I argue that such a transition—implementing law and policies in software and relying on them solely—is a potentially dangerous one. Such replacements should be carefully designed to include the flexible architecture already present in existing modes of policy interpretation and adjudication, lest it be carelessly discarded in favor of scalability, simplicity or cost. Without the benefit of reasonable interpretation, codified versions of the law will be left vulnerable to exploitation by attackers and criminals, and may ensnare innocent citizens in an overly literal or buggy interpretation.

Moving to automated and computer-driven enforcement of laws appears to be the natural next step in the progression of our technophilic society. Computers' ability to be told how to do something, and then perform it faster, cheaper, and more efficiently by several orders of magnitude than ever before imagined is an intoxicating power. This power has been put to great use, helping to improve our understanding in and across nearly every field imaginable. Computers have been used in the engineering and design fields to improve structural safety and lower costs of buildings and bridges. They have been used to simulate the complicated inner-workings of life itself to improve our understanding of biology. They help directly control a great number of systems from the tedium of welding and assembly robots in factories to impressively death-defying systems such as aircraft and rocket controls, many of which are simply too fast or unstable to trust to human operators. When it comes to automating our society with computers, the question is not whether or not computers should be employed, but rather when the technology will be ready to replace the human counterpart.

Of course, the drive to automation is not unique to computers. Take for example the farmer Frank W. Andrew. In 1940, Frank found a clever way to automate plowing and planting with a tractor. He attached a cable to the front axle of a tractor, with the other end wound around a barrel mounted in the middle of the field. When the tractor was driven, the wire would wind around the barrel, and keep the tractor in smaller and smaller spirals toward the center [2]. Frank had invented a crude form of driverless tractor, decades before the first ideas of having computers drive them. Of course, today, several versions of computer-driven tractors are commercially available from major manufacturers, and self-driving cars are on the verge of becoming mainstream on our roads.

Given our propensity toward automation, combined with the increasingly useful and constantly improving tool that computers provide, it seems inevitable that soon everything will be controlled by computers in one form or another. Certainly many open problems are difficult from a technical standpoint to implement in software, such as arbitrary object recognition [144, 90] or natural language processing [21]. But these hurdles are accompanied by a sea of other problems that were previously thought difficult or impossible to solve, but are now more or less surmounted. Computers can now extract text from images nearly as well as humans [62]. They can transcribe speech from audio for many languages [63] (and often translate between them). Algorithms—and associated smartphone apps—can listen to a song and recognize it, telling you the song name, artist, album, and even a list of other songs that are similar in genre or style [96]. These impressive accomplishments suggest that implementing any system in software is a matter of will, and that eventually most if not all domains will be run, handled, or monitored by computers.

But are there domains where this type of automation could be problematic? Not in the sense of technically difficult or blocked by unsolved “open problems”, but in the sense that if we do manage to implement them in hardware and software, they might cause negative externalities to the users or even to society at large. In this thesis, I argue that public policy and law is one such domain. In particular, policies that face *adversarial pressure*—that is, where there exist people who stand to gain by exploiting or circumventing the policies—can have negative external consequences from computer implementation.

Computable Contracts As an example, consider the relatively new but evolving area of computable contracts. In a computable contract, an agreement between parties is codified not in a written English document but rather in a series of conditions and data points, so that a computer program can automatically arbitrate the terms of the agreement [143]. For instance, imagine Alice agrees to pay Bob by a certain date or else forfeit some collateral. Alice and Bob could agree to use a computable contract, where an automated escrow service monitors and enforces this obligation. This escrow is programmed to watch money transfers between Alice and Bob, and checks to see if the money Alice sends Bob meets the agreed amount by the certain date. If Alice satisfies this condition, the escrow pays back her collateral; otherwise, it is given to Bob.

Suppose that Alice is malicious and wants to defraud Bob of both the collateral and the agreed money. One trick that Alice might try is to take out a second loan from Bob, and immediately pay it back in a way that is observable to the automated escrow service. Without additional context, the escrow service might be tricked into believing this payment satisfies the original contractual obligation of Alice to pay Bob, triggering the condition to pay the collateral back to Alice. This is a bug in the automated escrow, and one that might even been triggered by accident if Alice and Bob perform other frequent transactions. But this kind of vulnerability seems obvious only in hindsight. When the escrow was originally written, the programmer might not have considered what happens when Alice and Bob have multiple simultaneous transactions; if they had, they would have put additional checks or clarifications in the code to account for this.

Similarly, the authors of the U.S. Code were not at the time of writing it thinking of how the word “use” could be construed to mean “employed as a monetary instrument” in a drug trafficking crime. In that case, the Supreme Court had to intervene to determine the correct interpretation. However, the deterministic computer that executes the instructions to carry out an automated escrow does not understand the intent behind the contract (at least, not in the way that the Supreme Court is expected to interpret the intent behind the U.S. Code). Because there is an adversary (Alice) willing to exploit the failure of the machine to accurately enforce the intent of the rules, the system causes money to go to the wrong parties. And although future audits could reveal this kind of oversight and retroactively punish Alice for her misdeed, unless this audit is somehow part of the standard process there will always be a chance that such a system will fail to enforce the rules as intended.

High Frequency Trading In recent decades, the stock market transformed from a system where human brokers handled trades to one where computers rapidly executed and issued the majority of transactions automatically. Computers could react faster than human traders by orders of magnitude, provided that the computers could be programmed to interpret relevant information in a way that gave a clear decisive trading action. One example of this is arbitrage, where if there are multiple exchanges that trade the same commodity (or one trades derivatives of the commodity), there may be times when the prices between exchanges diverge enough to allow an instant profit. For example, if a sell order for a commodity is placed on the Chicago Mercantile Exchange, selling at a value below a buy order for the same commodity at the New York Stock Exchange, a quick profit can be made by buying the stock in Chicago and selling it in New York. Such an arbitrage will not exist for long, and the first person (or computer) to act on it will make a profit of the difference between the prices (minus any trading fees). Such an easy profit is essentially a race, and computers that have fast processing and low-latency connections between Chicago and New York are at a significant advantage to winning this race. Indeed, in 2009, a company called Spread Networks spent $300 million on laying a straighter fiber optic cable through the Allegheny mountains in order to cut the latency between Chicago and New York by approximately 2 milliseconds [95].

The tactic of using computers to trade at these blazing speeds is termed *high frequency trading*, and has drastically changed the way and strategy of trading stocks [95]. However, there are questions raised over the ethical and legal status of this style of trading [27]. In the early days of high frequency trading, humans still traded alongside the computers, often blindly unaware of the algorithms that watched them [95]. This allowed computers with especially low-latency connections to observe human-placed orders in one exchange, and front-run them in another: by buying the rest of the human trader's order first, the computer trading algorithm would profit from forcing the human to spend more to complete their buy order.

This type of trading is potentially predatory, and many have called to ban or discourage it, either through regulation or technical means [161]. Part of the challenge to this problem of this race to the bottom is that the system as it was built—the rules, the software, and protocols—allowed it technically, even though it did not originally intend it. This kind of unintended incentive shift has had a dramatic impact on financial markets and society, including an invisible tax on all other kinds of trading.

In 2010, however, the high-frequency traders had a comeuppance of sorts: on May 6, several stock indexes rapidly crashed, and then nearly as quickly rebounded. In the span of about half an hour, over a trillion dollars was wiped off the markets [124]. Investors, traders, and regulators were initially baffled as to what could have caused such rapid instability. While it is still not certain what event (or events) triggered the crash, a British trader named Navinder Singh Sarao has been charged with several counts of fraud and market manipulation [24] that may have led to the so called "flash crash". Sarao is alleged to have spoofed sell orders to the market, which he then immediately canceled [43]. This caused high frequency trading algorithms to attempt to act on the non-existent orders, also selling large quantities of shares themselves, leading to the rapid crash and market instability.

Both the predatory trading as well as the flash crash illustrate the risk that moving enforcement of rules to automated computer processes can introduce, particularly when there are competitive organizations willing to engage in adversarial behavior in order to profit. To be clear, there are many reasons that contribute to this type of predatory trading existing, but having the rules interpreted blindly by computers, with little human oversight is one contributing factor. Without this, it would have been harder for traders to reliably create algorithms that would consistently beat the competition, without fear of it being discovered by a human observer. Here it isn't so much that there was a bug in the implementation of the stock market (though such a bug could be equally if not more devastating), but that by hiding the process from eyes that knew what behavior was supposed to happen, exploitation of the system could happen virtually invisibly.

Code is Law When the Internet became more accessible, many people believed this new space would be a revolution of sorts, that it would be free from government regulation or corruption, and that it would stand to resist these forces forever. In the early days, some of this was certainly true: users were relatively anonymous by simply using screen names, and governments often did not even understand how the Internet worked or what it could be used for, let alone how to regulate or influence it [149].

However, Lawrence Lessig argues that the Internet is not as free from this type of regulation as it appeared [93, 94]. Certainly it would take time for governments to catch up, but he argued that there wasn't anything intrinsic to the Internet that made it resistant to control or co-option by governments. In fact, he argued just the opposite was true: on the Internet, there is a new form of law, which is the code that runs in the software and hardware of routers and end computers. In its early versions, this code was written by hobbyists and researchers, and so was instilled with the kinds of laws and controls that those types of people would want: free, open, and with minimal regulation or impediments to users of the system. However, this code could still be changed, co-opted, regulated, or influenced by governments. And by doing this, governments could regain control over the "law" of the Internet, including what users could do online, how they were tracked, or even how they behaved in online communities. Lessig argued that *code is law*, and that this new form of power wasn't impervious to the same kinds of forces governments used to change and influence traditional forms of law and power before.

Law is Code This thesis looks at this in the opposite direction. What happens when existing legal structures or public policies are implemented in code? If code is law as Lessig writes, then it seems law (and its enforcement) can also be made into code. We can choose to transfer the power currently granted to law and policies as they are written to computers that implement and enforce them. But what are the consequences to such a transition? This dissertation argues that law implemented as code lacks the flexibility and interpretability that is so important to our society. When computers rigidly and blindly interpret our legal and regulatory structures, any typos, bugs, or vulnerabilities can lead to attackers finding ways to circumvent the intended policy.

To look at the consequences of implementing policies in more technical detail, this dissertation investigates three distinct examples of public policies being implemented in code, and discovers the vulnerabilities and difficulties that lie beneath the surface of these systems:

- **Internet Voting** In an effort to increase voter turnout and decrease the cost of elections, many governments are moving toward electronic or Internet-based voting systems [5, 6]. Here, governments are taking a policy of how votes are collected and counted, and implementing it entirely in software and hardware. This technological shift potentially brings with it several vulnerabilities, many of which are unique to the domain of voting. By studying a real-world Internet voting system [171], we illustrate how such a system fails in practice, and suggest systemic remedies that system designers can adopt to make these systems harder to attack.

- **Airport Security** Prompted by (foiled) terrorist plots to destroy commercial airlines with plastic explosives, the Transportation and Security Administration (TSA) has deployed new Advanced Imaging Technology (AIT) machines with the goal of detecting these non-metallic threats. In the form of full-body scanners, these machines automatically capture images of passengers using either X-ray backscatter or millimeter wave detectors in order to look for anomalous contraband. To study this, we dissect and evaluate an X-ray backscatter machine similar to the ones deployed at airports until 2013. We find that despite their stated purpose being to detect plastic explosives, these machines can be circumvented with a little knowledge of their operation. We are able to conceal both firearms and radiological simulants of plastic explosive using simple techniques [105]. We also find a privacy side-channel that allows other passengers to potentially reconstruct the sensitive naked images produced by these machines remotely. These vulnerabilities are by no means intentional, but they represent a distinct risk of using these machines. Moreover, they illustrate and underscore the risks of implementing such checks in hardware and software.

- **Internet Censorship** Numerous governments attempt to censor the information that their citizens can access [148]. While this censorship was previously enacted by the government controlling newspapers and television content publishers [109], today these governments must now adapt to an Internet-connected world. One step countries such as China and Iran have taken is to implement their censorship rules in code, deploying censoring firewalls and routers that attempt to enforce their policy to Internet traffic [178, 19]. Although there is a known cat-and-mouse game between censors and free-speech advocates, researchers and activists attempting to circumvent these firewalls [52, 28, 74, 69, 169, 163, 103, 165, 56], the censors remain vigilant and often have an advantage in resources or scale that continues to hamper the free flow of information [49]. In this study, we describe a fundamentally new approach to anticensorship, that attempts to level the playing field [175, 174]. By placing proxies at Internet Service Providers (ISPs) outside the censoring country, censors will be unable to block access to them without unduly disconnecting their users from large otherwise legitimate portions of the Internet. This again illustrates the brittleness of implementing enforcement of public policy in automatic rules. In this case, the "adversary" who benefits is the ordinary citizen that wishes to access content and information freely. However, policy and law implemented as code is at a high risk of suffering this kind of problem: any vulnerabilities, weaknesses or mistakes can be leveraged to circumvent the system entirely.

Moving Forward Given the dangers of implementing public policy and law in code, what can be done? After all, we cannot expect or ask that innovation and progress be halted until we know how to better secure high-risk implementations. Rather than simply point to problems and say that implementations of public policy or law should be forbidden, we should look for ways in which we can use as much automation as possible while still remaining robust to security failures.

Adding redundancies and human-auditable logs into software-only checks could make it harder for attackers (or chance) to exploit programmer error. For example, ballots recorded electronically could keep auditable paper-based records of the ballots, such as in precinct count optical scan systems [57]. Ultimately recognizing where these weaknesses lie in a given system will likely require adversarial thinking or a "security mindset". There is unlikely to be a complete or general solution, but by keeping security in mind during the design and deployment of systems, and avoiding automation when the risks are too high, we can develop reasonably secure systems that modern society can depend on.

CHAPTER 2

Internet Voting

2.1 Introduction

Conducting elections for public office over the Internet raises grave security risks. A web-based voting system needs to maintain both the integrity of the election result and the secrecy of voters' choices, it must remain available and uncompromised on an open network, and it has to serve voters connecting from untrusted clients. Traditional elections caried out on paper ballots naturally have a recorded audit log, and the process for collecting these ballots is both simple and easy for the minimally-trained election volunteers and officials to understand. These human volunteers are able to intuit seemingly obvious assumptions that protect the security of the election. For example, each voter should only submit one ballot, and only specific election officials should be allowed to take the ballot box (and only after the poll has closed).

When these interpreting volunteers are replaced with complicated software and code that is blindly executed by computers, such unspoken yet obvious rules can be hard to fully capture. This can lead to vulnerabilities that allow attackers to manipulate the outcome of an election. Furthermore, many Internet voting designs centralize or aggregate the vote storing and tallying in a central server, exacerbating the risk (and reward) of attack.

Indeed, many security researchers have already cataloged threats to Internet voting (e.g. [77, 128]), even as others have proposed systems and protocols that may be steps to solutions someday (e.g. [11, 85]); meanwhile, a growing number of states and countries have been charging ahead with systems to collect votes online. Estonia [5] and Switzerland [6] have already adopted online voting for national elections. As of 2010, 19 U.S. states employed some form of Internet voting [9], and at least 12 more were reportedly considering adopting it [8].

Among the jurisdictions considering Internet voting, one of the most enthusiastic proponents was the District of Columbia. In 2010, the Washington, D.C. Board of Elections and Ethics (BOEE) embarked on a Federally-funded pilot project that sought to allow overseas voters registered in the District to vote over the web starting with the November 2010 general election [141]. Though the D.C. system, officially known as the “D.C. Digital Vote-by-Mail Service,” was technologically similar to parallel efforts in other states, BOEE officials adopted a unique and laudable level of transparency. The system was developed as an open source project, in partnership with the nonprofit Open Source Digital Voting (OSDV) Foundation [7]. Most significantly, prior to collecting real votes with the system, the District chose to operate a mock election and allow members of the public to test its functionality and security.

We participated in this test, which ran for four days in September and October 2010. Our objective was to approach the system as real attackers would: starting from publicly available information, we looked for weaknesses that would allow us to seize control, unmask secret ballots, and alter the outcome of the mock election. Our simulated attack succeeded at each of these goals and prompted the D.C. BOEE to discontinue its plans to deploy digital ballot return in the November election.

In this chapter, we provide a case study of the security of an Internet voting system that, absent our participation, might have been deployed in real elections. Though some prior investigations have analyzed the security of proposed Internet voting systems by reviewing their designs or source code, this is the first instance of which we are aware where researchers have been permitted to attempt attacks on such a system in a realistic deployment intended for use in a general election.

We hope our experiences with the D.C. system will aid future research on secure Internet voting. In particular, we address several little-understood practical aspects of the problem, including the exploitability of implementation errors in carefully developed systems and the ability of election officials to detect, respond, and recover from attacks. Our successful penetration supports the widely held view among security researchers that web-based electronic voting faces high risks of vulnerability, and it cautions against the position of many vendors and election officials who claim that the technology can readily be made safe.

The remainder of this chapter is organized as follows: Section 2.2 introduces the architecture and user interface of the Digital Vote-By-Mail System. In Section 2.3, we describe how we found and exploited vulnerabilities in the web application software to compromise the mock election. Section 2.4 describes further vulnerabilities that we found and exploited in low-level network components. Section 3.5 discusses implications of our case study for other Internet voting systems and future public trials, and we conclude in Section 2.7.

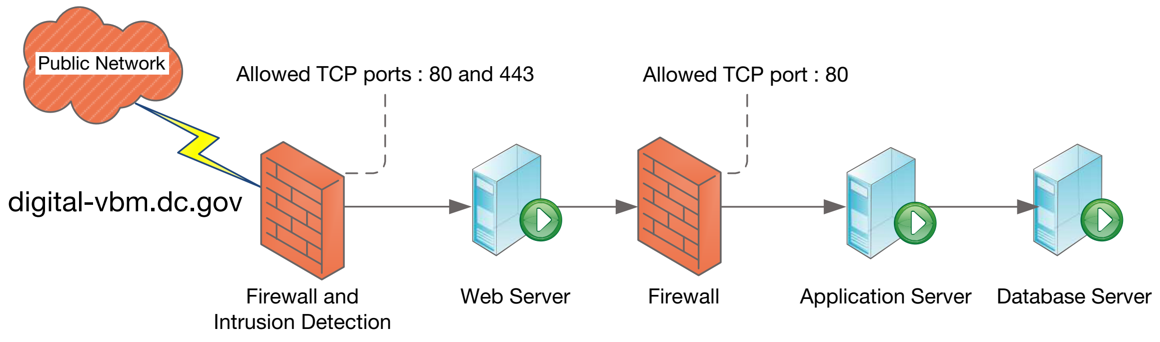


Figure 2.1: **Network architecture** — The front-end web server receives HTTPS requests from users and reverse-proxies them to the application server, which hosts the DVBM election software and stores both blank and completed ballots. A MySQL database server stores voter credentials and tracks voted ballots. Multiple firewalls reduce the attack surface and complicate attacks by disallowing outbound TCP connections. The intrusion detection system in front of the web server proved ineffective, as it was unable to decrypt the HTTPS connections that carried our exploit. (Adapted from `http://www.dcboee.us/DVM/Visio-BOEE.pdf`.)

The material in this chapter is adapted from "Attacking the Washington, DC Internet voting system" by Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman, which originally appeared in *Financial Cryptography and Data Security*, February 2012.

2.2 Background: The D.C. Digital Vote-By-Mail System

Architecture The Digital Vote-by-Mail (DVBM) system is built around an open-source web application[1] developed in partnership with the D.C. BOEE by the OSDV Foundation's TrustTheVote project[2]. The software uses the popular Ruby on Rails framework and is hosted on top of the Apache web server and the MySQL relational database. Global election state (such as registered voters' names, addresses, hashed credentials, and precinct-ballot mappings, as well as which voters have voted) is stored in the MySQL database. Voted ballots are encrypted and stored in the filesystem. User session state, including the user ID and whether the ballot being cast is digital or physical, is stored in an encrypted session cookie on the user's browser.

Electronic ballots are served as PDF files which voters fill out using a PDF reader and upload back to the server. To safeguard ballot secrecy, the server encrypts completed ballots with a public key whose corresponding private key is held offline by voting officials. Encrypted ballots are stored on the server until after the election, when officials transfer them

[1] `http://github.com/trustthevote/DCdigitalVBM/`
[2] `http://trustthevote.org`

to a non-networked computer (the “crypto workstation”), decrypt them using the private key, and print them for counting alongside mail-in absentee ballots.

Figure 2.1 shows the network architecture deployed for the mock election. HTTPS web requests are interpreted by the web server over TCP port 443. The web server then performs the HTTP request on the user’s behalf to the application server, which runs the DVBM application software. The web server, application server, and a MySQL database server all run Linux. Firewalls prevent outbound connections from the web and application servers. Since the web server and application server run on separate machines, a compromise of the application server will not by itself allow an attacker to steal the HTTPS private key.

Voter experience The DVBM system was intended to be available to all military and overseas voters registered in the District. Months prior to the election, each eligible voter received a letter by postal mail containing credentials for the system. These credentials contained the voter ID number, registered name, residence ZIP code, and a 16-character hexadecimal personal identification number (PIN). The letters instructed voters to visit the D.C. Internet voting system website, which guided them through the voting process.

Upon arrival, the voter selects between a digital or postal ballot return. Next, the voter is presented with an overview of the voting process. The voter then logs in with the credentials provided in the mail, and confirms his or her identity. Next, the voter is presented with a blank ballot in PDF format. In the postal return option, the voter simply prints out the ballot, marks it, and mails it to the provided address. For the digital return, the voter marks the ballot electronically using a PDF reader, and saves the ballot to his or her computer. The voter then uploads the marked ballot to the D.C. Internet voting system, which reports that the vote has been recorded by displaying a “Thank You” page. If voters try to log in a second time to cast another ballot, they are redirected to the final Thank You page, disallowing them from voting again.

2.3 Attacking the Web Application

In this section, we describe vulnerabilities we discovered and exploited in the DVBM server application. Our search for vulnerabilities was primarily conducted by manual inspection of the web application’s source code, guided by a focus on the application’s attack surface. In particular, we concentrated on voter login, ballot upload and handling, database communication, and other network activity. The fact that the application was open source expedited our search, but motivated attackers could have found vulnerabilities without the source code using alternative methods. For example, one might attack voter login fields, ballot contents, ballot filenames, or session cookies, by either fuzzing or more direct code injection attacks such as embedding snippets of SQL, shell commands, and popular scripting languages with detectable side effects.

2.3.1 Shell-injection vulnerability

After a few hours of examination, we found a shell injection vulnerability that eventually allowed us to compromise the web application server. The vulnerability was located in the code for encrypting voted ballots uploaded by users. The server stores uploaded ballots in a temporary location on disk, and the DVBM application executes the `gpg` command to encrypt the file, using the following code:

```
run("gpg", "--trust-model always -o \"#{File.expand_path(dst.path)}\"
    -e -r \"#{@recipient}\" \"#{File.expand_path(src.path)}\"")
```

The `run` method invoked by this code concatenates its first and second arguments, collapses multiple whitespace characters into single characters, and then executes the command string using Ruby's backtick operator, which passes the provided command to the shell. The Paperclip[3] Rails plugin, which the application uses to handle file uploads, preserves the extension of the uploaded ballot file, and no filtering is performed on this extension, so the result of `File.expand_path(src.path)` is attacker controlled. Unfortunately, in the Bash shell used on the server, double quotes do not prevent the evaluation of shell metacharacters, and so a ballot named `foo.$(cmd)` will result in the execution of `cmd` with the privileges of the web application.

The current release of the Paperclip plugin at the time of our analysis (late September 2010) was version 2.3.3. It appears that a similar vulnerability in Paperclip's built-in `run` method was fixed on April 30, 2010[4]. The first release containing the patch was version 2.3.2, which was tagged in the Paperclip Git repository on June 8, 2010. The degree of similarity between the DVBM application's custom `run` method and the Paperclip `run` method suggests that the DVBM application's implementation is a custom "stripped-down" version of Paperclip's, contrary to the D.C. BOEE's assertion that "a new version of [Paperclip] that had not been fully tested had been released and included in the deployed software" and "did not perform filename checks as expected." [126] Indeed, if DVBM had used the Paperclip `run` method together with an up-to-date version of the Paperclip library, this specific vulnerability would not have been included in the software. The resulting attack serves as a reminder that a small, seemingly minor engineering mistake in practically any layer of the software stack can result in total system compromise.

[3] https://github.com/thoughtbot/paperclip

[4] The patch in question is available at https://github.com/thoughtbot/paperclip/commit/724cc7. It modifies `run` to properly quote its arguments using single quotes.

When we tested the shell injection vulnerability on the mock election server, we discovered that outbound network traffic from the test system was filtered, rendering traditional shellcode and exfiltration attempts (e.g., `nc umich.edu 1234 < /tmp/ballot.pdf`) ineffective. However, we were able to exfiltrate data by writing output to the `images` directory on the compromised server, where it could be retrieved with any HTTP client. To expedite crafting our shell commands, we developed an exploit compiler and a shell-like interface that, on each command, creates a maliciously named ballot file, submits the ballot to the victim server, and retrieves the output from its chosen URL under `/images`.

Interestingly, although the DVBM system included an intrusion detection system (IDS) device, it was deployed in front of the web server and was not configured to intercept and monitor the contents of the encrypted HTTPS connections that carried our attack. Although configuring the IDS with the necessary TLS certificates would no doubt have been labor intensive, failure to do so resulted in a large "blind spot" for the D.C. system administrators.

2.3.2 Attack payloads

We exploited the shell injection vulnerability to carry out several attacks that illustrate the devastating effects attackers could have during a real election if they gained a similar level of access:

Stealing secrets We retrieved several cryptographic secrets from the application server, including the public key used for encrypting ballots. Despite the use of the term "public key," this key should actually be kept secret, since it allows attackers to substitute arbitrary ballots in place of actual cast ballots should they gain access to the storage device. We also gained access to the database by finding credentials in the bash history file (`mysql -h 10.1.143.75 -udvbm -pP@ssw0rd`).

Changing past and future votes We used the stolen public key to replace all of the encrypted ballot files on the server at the time of our intrusion with a forged ballot of our choosing. In addition, we modified the ballot-processing function to append any subsequently voted ballots to a `.tar` file in the publicly accessible `images` directory (where we could later retrieve them) and replace the originals with our forged ballot. Recovery from this attack is difficult; there is little hope for protecting future ballots from this level of compromise, since the code that processes the ballots is itself suspect. Using backups to ensure that compromises are not able to affect ballots cast prior to the compromise may conflict with ballot secrecy in the event that the backup itself is compromised.

Revealing past and future votes One of the main goals of a voting system is to protect ballot secrecy, which means not only preventing an attacker of the system from determining how a voter voted, but also preventing a voter from willingly revealing their cast ballot to a third party, even if they are coerced or incentivized to do so. While any absentee system that allows voters to vote where they choose allows a voter to reveal his or her vote voluntarily, our attack on the D.C. system allowed us to violate ballot secrecy and determine how nearly all voters voted.

Our modifications to the ballot processing function allowed us to learn the contents of ballots cast following our intrusion. Revealing ballots cast prior to our intrusion was more difficult, because the system was designed to store these ballots in encrypted form, and we did not have the private key needed to decipher them. However, we found that the Paperclip Rails plugin used to handle file uploads stored each ballot file in the `/tmp` directory before it was encrypted. The web application did not remove these unencrypted files, allowing us to recover them. While these ballots do not explicitly specify the voter's ID, they do indicate the precinct and time of voting, and we were able to associate them with voters by using login events and ballot filenames recorded in the server application logs. Thus, we could violate the secret ballot for past and future voters.

Discovering that real voter credentials were exposed In addition to decrypted ballots, we noticed that the `/tmp` directory also contained uploaded files that were not PDF ballots but other kinds of files apparently used to exercise error handling code during testing. To our surprise, one of these files was a 937 page PDF document that contained the instruction letters sent to each of the registered voters, which included the real voters' credentials for using the system. These credentials would have allowed us (or anyone else who penetrated the insecure server) to cast votes as these citizens in the real D.C. election that was to begin only days after the test period. Since the system requires that these credentials be delivered via postal mail, it would be infeasible for officials to send updated ones to the voters in time for the election.

Hiding our tracks We were able to hide the evidence of our intrusion with moderate success. We downloaded the DVBM application logs, altered them to remove entries corresponding to our malicious ballot uploads, and, as our final actions, overwrote the application log with our sanitized version and removed our uploaded files from the `/tmp` and `images` directories.

Our calling card To make our control over the voting system more tangible to nontechnical users, we left a "calling card" on the final screen of the digital voting workflow: we uploaded a recording of "The Victors" (the University of Michigan fight song) and modified the confirmation page to play this recording after several seconds had elapsed. We hoped that this would serve as a clear demonstration that the site had been compromised, while remaining discreet enough to allow the D.C. BOEE system administrators a chance to exercise their intrusion detection and response procedures.

2.3.3 Other vulnerabilities and potential attacks

Our intention in participating in the trial was to play the role of a real attacker. Therefore, once we had found vulnerabilities that allowed us to compromise the system, our attention shifted to understanding and exploiting these problems. However, along the way we did uncover several additional vulnerabilities in the DVBM web application that were not necessary for our attack. Two key system deployment tasks were not completed. First, the set of test voter credentials was not regenerated and was identical to those included in the public DVBM Git repository. While the test voter credentials were fictitious, their disclosure constituted a security problem because public testers were asked to contact the D.C. BOEE for credentials, implying that the number of credentials available to each test group was to be limited.

Similarly, the encryption key used for session cookies was unchanged from the default key published in the repository. Disclosure of the key exacerbated a second vulnerability: rather than using the Rails-provided random `session_id` to associate browser sessions with voter credentials, the DVBM developers used the `rid` value, which corresponds to the automatically incremented primary key of the registration table in the system's MySQL database. This means every integer less than or equal to the number of registered voters is guaranteed to correspond to some voter. Combining this with the known encryption key results in a *session forgery* vulnerability. An attacker can construct a valid cookie for some voter simply by choosing an arbitrary valid `rid` value. This vulnerability could have been used to submit a ballot for every voter.

Our attack was expedited because the DVBM application user had permission to write the code of the web application. Without this permission, we would have had to find and exploit a local privilege escalation vulnerability in order to make malicious changes to the application. However, as we were able to carry out our attacks as the web application user, we did not need to find or use such an exploit.

We also identified other attack strategies that we ultimately did not need to pursue. For instance, the "crypto workstation" (see Section 2.2) used for decrypting and tabulating ballots is not directly connected to the Internet, but attackers may be able to compromise it by exploiting vulnerabilities in PDF processing software. PDF readers are notoriously subject to security vulnerabilities; indeed, the Crypto Workstation's lack of Internet connectivity may reduce its security by delaying the application of automated updates in the time leading up to the count. If the Crypto Workstation is compromised, attackers would likely be able to rewrite ballots. Furthermore, the web application allowed uploaded PDF ballots to contain multiple pages. If the printing is done in an automated fashion without restricting printouts to a single page, an attacker could vote multiple ballots.

2.4 Attacking the Network Infrastructure

In addition to the web application server, we were also able to compromise network infrastructure on the pilot network. This attack was independent from our web application compromise, yet it still had serious ramifications for the real election and showed a second potential path into the system.

Prior to the start of the mock election, the D.C. BOEE released a pilot network design diagram that showed specific server models, the network configuration connecting these servers to the Internet, and a CIDR network block (8.15.195.0/26). Using Nmap, we discovered five of the possible 64 addresses in this address block to be responsive. By using Nmap's OS fingerprinting feature and manually following up with a web browser, we were able to discover a Cisco router (8.15.195.1), a Cisco VPN gateway (8.15.195.4), two networked webcams (8.15.195.11 and 8.15.195.12), and a Digi Passport 8 terminal server[5] (8.15.195.8).

2.4.1 Infiltrating the terminal server

The Digi Passport 8 terminal server provides an HTTP-based administrative interface. We were able to gain access using the default root password (`dbps`) obtained from an online copy of the user manual. We found that the terminal server was connected to four enterprise-class Cisco switches (which we surmised corresponded to the switches shown on the network diagram provided by the BOEE) and provided access to the switches' serial console configuration interfaces via telnet.

We hid our presence in the terminal server using a custom JavaScript rootkit, which we installed over an SSH session (the same account names and passwords used in the web interface were accepted for SSH). The rootkit concealed an additional account with administrator privileges, "dev," which we planned to use in case our attack was discovered and the passwords changed. We also used our SSH access to download the terminal server's `/etc/shadow` and `/etc/passwd` files for cracking using the "John the Ripper"

[5]A *terminal server* is a device that attaches to other pieces of equipment and allows administrators to remotely log in and configure them.

password cracker[6]. After about 3.5 hours using the cracker's default settings, we recovered the secondary administrator password `cisco123` from a salted MD5 hash.

Evidence of other attackers When we inspected the terminal server's logs, we noticed that several other attackers were attempting to guess the SSH login passwords. Such attacks are widespread on the Internet, and we believe the ones we observed were not intentionally directed against the D.C. voting system. However, they provide a reminder of the hostile environment in which Internet voting applications must operate.

The first SSH attack we observed came from an IP address located in Iran (80.191.180.102), belonging to Persian Gulf University. We realized that one of the default logins to the terminal server (user: `admin`, password: `admin`) would likely be guessed by the attacker in a short period of time, and therefore decided to protect the device from further compromise that might interfere with the voting system test. We used iptables to block the offending IP addresses and changed the admin password to something much more difficult to guess. We later blocked similar attacks from IP addresses in New Jersey, India, and China.

2.4.2 Routers and switches

After we compromised the terminal server, we found several devices connected to its serial ports. Initially, there were four Cisco switches: a pair of Nexus 5010s and a pair of Nexus 7010s. Connecting to these serial ports through the terminal server presented us with the switches' login prompts, but previously found and default passwords were unsuccessful.

The terminal server provided built-in support for keystroke logging of serial console sessions and forwarding of logged keystrokes to a remote syslog server, which we enabled and configured to forward to one of our machines. This allowed us to observe in real time as system administrators logged in and configured the switches, and to capture the switches' administrative password, `!@#123abc`.

Later in the trial, four additional devices were attached to the terminal server, including a pair of Cisco ASR 9010 routers and a pair of Cisco 7606-series routers. We were again able to observe login sessions and capture passwords. At the end of the public trial, we changed the passwords on the routers and switches—effectively locking the administrators out of their own network—before alerting BOEE officials and giving them the new password.

D.C. officials later told us that the routers and switches we had infiltrated were not intended to be part of the voting system trial and were simply colocated with the DVBM servers at the District's off-site testing facility. They were, however, destined to be deployed in the core D.C. network, over which real election traffic would flow. With the access we had,

[6]`http://www.openwall.com/john/`

we could have modified the devices' firmware to install back doors that would have given us persistent access, then later programmed them to redirect Internet voting connections to a malicious server.

2.4.3 Network webcams

We found a pair of webcams on the DVBM network—both publicly accessible without any password—that showed views of the server room that housed the pilot. One camera pointed at the entrance to the room, and we were able to observe several people enter and leave, including a security guard, several officials, and IT staff installing new hardware. The second camera was directed at a rack of servers.

These webcams may have been intended to increase security by allowing remote surveillance of the server room, but in practice, since they were unsecured, they had the potential to leak information that would be extremely useful to attackers. Malicious intruders viewing the cameras could learn which server architectures were deployed, identify individuals with access to the facility in order to mount social engineering attacks, and learn the pattern of security patrols in the server room. We used them to gauge whether the network administrators had discovered our attacks—when they did, their body language became noticeably more agitated.

2.5 Discussion

2.5.1 Attack detection and recovery

After we completed our attack—including our musical calling card on the "Thank You" page—there was a delay of approximately 36 hours before election officials responded and took down the pilot servers for analysis. The attack was apparently brought to officials' attention by an email on a mailing list they monitored that curiously asked, "does anyone know what tune they play for successful voters?" Shortly after another mailing list participant recognized the music as "The Victors," officials abruptly suspended the public examination period, halting the tests five days sooner than scheduled, citing "usability issues."

Following the trial, we discussed the attack with D.C. officials. They explained that they found our modifications to the application code by comparing the disk image of the server to a previous snapshot, although this required several days of analysis. They confirmed that they were unable to see our attacks in their intrusion detection system logs, that they were unable to detect our presence in the network equipment until after the trial, and that they did not discover the attack until they noticed our intentional calling card. We believe that attack detection and recovery remain significant challenges for any Internet voting system.

2.5.2 Adversarial testing and mechanics of the D.C. trial

The D.C. BOEE should be commended for running a public test of their system. Their trial was a step in the right direction toward transparency in voting technology and one of the first of its kind. Nonetheless, we reiterate that adversarial testing of Internet voting applications is not necessary to show that they are likely to be weak. The architectural flaws inherent in Internet voting systems in general and the potential disastrous implications of a single vulnerability were known and expected by researchers prior to the D.C. trial [77]. We hope not to have to repeat this case study in order to highlight these limitations once again.

The key drawback to adversarial testing is that a lack of problems found in testing *does not* imply a lack of problems in the system, despite popular perception to the contrary. It is likely that testers will have more limited resources and weaker incentives than real attackers—or they may simply be less lucky. A scarcity of testers also seems to have been an issue during the D.C. trial. During our compromise of the DVBM server, we were able to view the web access logs, which revealed only a handful of attack probes from other testers, and these were limited to simple failed SQL and XSS injection attempts.

One reason for the lack of participation may have been ambiguity over the legal protections provided to testers by the BOEE. Another possible reason is that the test began on short notice—the final start date was announced only three days in advance. If such a trial must be repeated, we hope that the schedule will be set well in advance, and that legal protections for participants will be strongly in place. In addition to the short notice, the scheduled conclusion of the test was only three days before the system was planned to be opened for use by real voters. Had the test outcome been less dramatic, election officials would have had insufficient time to thoroughly evaluate testers' findings.

Despite these problems, one of the strongest logistical aspects of the D.C. trial was that access to the code—and to some extent, the architecture—was available to the testers. While some observers have suggested that this gave us an unrealistic advantage while attacking the system, there are several reasons why such transparency makes for a more realistic test. Above and beyond the potential security benefits of open source code (pressure to produce better code, feedback from community, etc.), in practice it is difficult to prevent a motivated attacker from gaining access to source code. The code could have been leaked by the authors through an explicit sale by dishonest insiders, as a result of coercion, or through a compromised developer workstation. Since highly plausible attacks such as these are outside the scope of a research evaluation, it is not only fair but realistic to provide the code to the testers.

2.5.3 Why Internet voting is hard

Practical Internet voting designs tend to suffer from a number of fundamental difficulties, from engineering practice to inherent architectural flaws. We feel it is important to point them out again given the continued development of Internet voting systems.

Engineering practice Both the DVBM system and the earlier prototype Internet voting system SERVE [77] were built primarily on commercial-off-the-shelf (COTS) software (which, despite the use of the term "commercial," includes most everyday open-source software). Unfortunately, the primary security paradigm for COTS developers is still "penetrate and patch." While this approach is suitable for the economic and risk environment of typical home and business users, it is not appropriate for voting applications due to the severe consequences of failure.

Inherited DRE threats Relatively simple Internet voting systems like D.C.'s DVBM strongly resemble direct recording electronic (DRE) voting machines, in that there is no independent method for auditing cast ballots. If the voting system software is corrupt, recovery is likely to be impossible, and even detection can be extremely difficult. DRE voting is highly susceptible to insider attacks as well as external compromise through security vulnerabilities. In previous work [18, 29, 54, 86, 170], the closed, proprietary nature of DREs has been held as an additional threat to security, since there is no guarantee that even the *intended* code is honest and correct. In contrast, the DVBM system was open source, but the public would have had no guarantee that the deployed voting system was actually running the published code.

Tensions between ballot secrecy and integrity One of the fundamental reasons that voting systems are hard to develop is that two fundamental goals of a secret ballot election—ballot secrecy and ballot integrity—are in tension. Indeed, the D.C. system attempted to protect integrity through the use of logs, backups and intrusion detection, yet these systems can help an intruder compromise ballot secrecy. Other security mechanisms put in place to protect ballot secrecy, such as encrypting completed ballots and avoiding incremental backups make detecting and responding to compromise much more difficult.

Architectural brittleness in web applications The main vulnerability we exploited resulted from a tiny oversight in a single line of code and could have been prevented by using single quotes instead of double quotes. Mistakes like this are all too common. They are also extremely hard to eradicate, not because of their complexity, but because of the multitude of potential places they can exist. If any one place is overlooked, an attacker may be able to leverage it to gain control of the entire system. In this sense, existing web application frameworks tend to be *brittle*. As our case study shows, the wrong choice of which type of quote to use—or countless other seemingly trivial errors—can result in an attacker controlling the outcome of an election.

Internet-based threats Internet voting exposes what might otherwise be a small, local race of little global significance to attackers from around the globe, who may act for a wide range of reasons varying from politics to financial gain to sheer malice. In addition to compromising the central voting server as we did, attackers can launch denial-of-service attacks aimed at disrupting the election, they can redirect voters to fake voting sites, and they can conduct widespread attacks on voters' client machines [51]. These threats correspond to some of the most difficult unsolved problems in Internet security and are unlikely to be overcome soon.

Comparison to online banking While Internet-based financial applications, such as online banking, share some of the threats faced by Internet voting, there is a fundamental difference in ability to deal with compromises after they have occurred. In the case of online banking, transaction records, statements, and multiple logs allow customers to detect specific fraudulent transactions and in many cases allow the bank to reverse them. Internet voting systems cannot keep such fine-grained transaction logs without violating ballot secrecy for voters. Even with these protections in place, banks suffer a significant amount of online fraud but write it off as part of the cost of doing business; fraudulent election results cannot be so easily excused.

2.6 Related Work

Internet voting has been tried in various forms in several countries, including Australia, Estonia, France, Norway and the United States [160]. Previous studies on these systems have revealed vulnerabilities that could allow attackers that exploit them to change votes or reveal voters' secret ballots.

One such example is the 2004 security analysis of the Secure Electronic Registration and Voting Experiment (SERVE) by Jefferson et al. [77]. SERVE was an Internet voting "pilot" that was slated for use in an actual election by absentee overseas voters. Jefferson et al. reviewed the system design and pointed out many architectural and conceptual weaknesses that apply to remote Internet voting systems in general, though they did not have an opportunity to conduct a penetration test of a pilot system. On the basis of these weaknesses, Jefferson et al. recommended "shutting down the development of SERVE immediately and not attempting anything like it in the future until both the Internet and the world's home computer infrastructure have been fundamentally redesigned." Despite incremental advances in computer security in the last eight years, the fundamental architectural flaws Jefferson et al. identified remain largely the same to this day.

More recently, Esteghari and Desmedt [51] developed an attack on the Helios 2.0 [11] open-audit Internet voting system. Their attack exploits an architectural weakness in home computer infrastructure by installing a "browser rootkit" or "man-in-the-browser attack" that detects the ballot web page and modifies votes. Esteghari and Desmedt note that Helios 3.0 is capable of posting audit information to an external web server *before* ballot submission, which can, in theory, be checked using a second trusted computer to detect the action of the rootkit, but it is not clear that such a second computer will be available or a sufficiently large number of nontechnical voters will take advantage of this audit mechanism.

Since 2005, Estonia has used Internet voting for its elections, making it the first country to do so. In 2014, Springall et al. studied the system and found several practical issues in the operational security in the initial setup of the system, including using a personal computer (with potential malware) to create the initial server images, revealing root passwords in published videos documenting the setup, and using previously used USB drives to transfer files [139].

In 2015, a review of the New South Wales, Australia iVote system revealed that a Javascript resource included on the voting page for tracking users was served from an HTTPS server vulnerable to the FREAK TLS attack, allowing a man-in-the-middle to modify or steal votes of citizens by replacing the Javascript with malicious code [64].

2.7 Conclusions

Our experience with the D.C. pilot system demonstrates one of the key dangers in many Internet voting designs: one small mistake in the configuration or implementation of the central voting servers or their surrounding network infrastructure can easily undermine the legitimacy of the entire election. We expect that other fielded Internet voting systems will fall prey to such problems, especially if they are developed using standard practices for mass-produced software and websites. Even if the central servers were somehow eliminated or made impervious to external attack, Internet voting is likely to be susceptible to numerous classes of threats, including sabotage from insiders and malware placed on client machines. The twin problems of building secure software affordably and preventing home computers from falling prey to malware attacks would both have to be solved before systems like D.C.'s could be seriously considered. This case study strongly illustrates the thesis of how the unforgiving nature of software bugs and literally-interpreted rules implemented in code can result in vulnerable elections. Securing Internet voting in practice will require significant fundamental advances in computer security, and we urge Internet voting proponents to reconsider deployment until and unless major breakthroughs are achieved.

CHAPTER 3

Airport Security

3.1 Introduction

In response to evolving terrorist threats, including non-metallic explosive devices and weapons, the U.S. Transportation Security Administration (TSA) has adopted advanced imaging technology (AIT), also known as whole-body imaging, as the primary passenger screening method at nearly 160 airports nationwide [154]. Introduced in 2009 and gradually deployed at a cost exceeding one billion dollars, AIT provides, according to the TSA, "the best opportunity to detect metallic and non-metallic anomalies concealed under clothing without the need to touch the passenger" [152].

AIT plays a critical role in transportation security, and decisions about its use are a matter of public interest. The technology comes as a more convenient and faster alternative for more invasive full-body pat-downs performed by human agents. However, questions remain whether checks for contraband can be automated safely and effectively. The technology has generated considerable controversy, including claims that the devices are unsafe [131], violate privacy and civil liberties [132, 97], and are ineffective [83, 34]. Furthermore, AIT devices are complex cyberphysical systems — much like cars [87] and implantable medical devices [58] — that raise novel computer security issues. Despite such concerns, neither the manufacturers nor the government agencies that deploy these machines have disclosed sufficient technical details to facilitate rigorous independent evaluation [131], on the grounds that such information could benefit attackers [152]. This lack of transparency has limited the ability of policymakers, experts, and the public to assess contradicting claims.

To help advance the public debate, we present the first experimental analysis of an AIT conducted independently of the manufacturer and its customers. We obtained a Rapiscan Secure 1000 full-body scanner — one of two AITs widely deployed by the TSA [111] — and performed a detailed security evaluation of its hardware and software. Our analysis provides both retrospective insights into the adequacy of the testing and evaluation procedures that

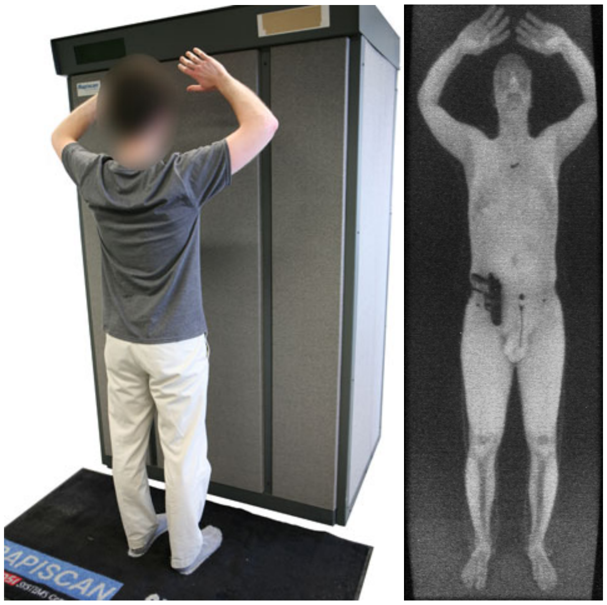


Figure 3.1: **The Rapiscan Secure 1000** full-body scanner uses backscattered X-rays to construct an image through clothing. Naïvely hidden contraband, such as the handgun tucked into this subject's waistband, is readily visible to the device operator.

led up to TSA use of the system, and prospective lessons about broader security concerns, including cyberphysical threats, that apply to both current and future AITs.

The Secure 1000 provides a unique opportunity to investigate the security implications of AITs in a manner that allows robust yet responsible public disclosure. Although it was used by the TSA from 2009 until 2013, it has recently been removed from U.S. airports due to changing functional requirements [115]. Moreover, while the Secure 1000 uses backscatter X-ray imaging, current TSA systems are based on a different technology, millimeter waves [38], so many of the attacks we present are not directly applicable to current TSA checkpoints, thus reducing the risk that our technical disclosures will inadvertently facilitate mass terrorism. However, while Secure 1000 units are no longer used in airports, they still are in use at other government facilities, such as courthouses and prisons (see, e.g., [71, 102]). In addition, other backscatter X-ray devices manufactured by American Science and Engineering are currently under consideration for use at airports [115]. To mitigate any residual risk, we have redacted a small number of sensitive details from our attacks in order to avoid providing recipes that would allow an attacker to reliably defeat the screening process without having access to a machine for testing.

In the first part of our study (Section 3.3), we test the Secure 1000's effectiveness as a physical security system by experimenting with different methods of concealing contraband. While the device performs well against naïve adversaries, fundamental limitations of backscatter imaging allow more clever attackers to defeat it. We show that an adaptive adversary, with the ability to refine his techniques based on experiment, can confidently smuggle contraband past the scanner by carefully arranging it on his body, obscuring it with other materials, or properly shaping it. Using these techniques, we are able to hide firearms, knives, plastic explosive simulants, and detonators in our tests. These attacks are surprisingly robust, and they suggest a failure on the part of the Secure 1000's designers and the TSA to adequately anticipate adaptive attackers. Fortunately, there are simple procedural changes that can reduce (though not eliminate) these threats, such as performing supplemental scans from the sides or additional screening with a magnetometer.

Next, we evaluate the security of the Secure 1000 as a cyberphysical system (Section 3.4) and experiment with three novel kinds of attacks against AITs that target their effectiveness, safety features, and privacy protections. We demonstrate how malware infecting the operator's console could selectively render contraband invisible upon receiving a "secret knock" from the attacker. We also attempt (with limited success) to use software-based attacks to bypass the scanner's safety interlocks and deliver an elevated radiation dose. Lastly, we show how an external device carried by the attacker with no access to the console can exploit a physical side-channel to capture naked images of the subject being scanned. These attacks are, in general, less practical than the techniques we demonstrate for hiding contraband, and their limitations highlight a series of conservative engineering choices by the system designers that should serve as positive examples for future AITs.

Finally, we attempt to draw broader lessons from these findings (Section 3.5). Our results suggest that while the Secure 1000 is effective against naïve attackers, it is not able to guarantee either efficacy or privacy when subject to attack by an attacker who is knowledgeable about its inner workings. While some of the detailed issues we describe are specific to the scanner model we tested, the root cause seems to be the failure of the system designers and deployers to think adversarially. This pattern is familiar to security researchers: past studies of voting machines [26], cars [87] and medical devices [58] have all revealed cyberphysical systems that functioned well under normal circumstances but were not secure in the face of attack. Thus, we believe this study reinforces the message that security systems must be subject to adversarial testing before they can be deemed adequate for widespread deployment.

Research safety and ethics. Since the Secure 1000 emits ionizing radiation, it poses a potential danger to the health of scan subjects, researchers, and passers by. Our institutional review board determined that our study did not require IRB approval; however, we worked closely with research affairs and radiation safety staff at the university that hosted our device to minimize any dangers and assure regulatory compliance. To protect passers by, our device was sited in a locked lab, far from the hallway, and facing a thick concrete wall. To protect researchers, we marked a 2 m region around the machine with tape; no one except the scan subject was allowed inside this region while high voltage was applied to the X-ray tube. We obtained a RANDO torso phantom [114], made from a material radiologically equivalent to soft tissue cast over a human skeleton, and used it in place of a human subject for all but the final confirmatory scans. For these final scans we decided, through consultation with our IRB, that only a PI would be used as a scan subject. Experiments involving weapons were conducted with university approval and in coordination with the campus police department and all firearms were unloaded and disabled. We disclosed our security-relevant findings and suggested procedural mitigations to Rapiscan and the Department of Homeland Security ahead of publication.

The material in this chapter is adapted from "Security Analysis of a Full-Body Scanner" by Keaton Mowery, Eric Wustrow, Tom Wypych, Cory Singleton, Chris Comfort, Eric Rescorla, Stephen Checkoway, J. Alex Halderman, and Hovav Shacham, which originally appeared in *Proceedings of the 23rd USENIX Security Symposium*, August 2014.

3.2 The Rapiscan Secure 1000

The Secure 1000 was initially developed in the early 1990s by inventor Steven W. Smith [137, 135]. In 1997, Rapiscan Systems acquired the technology [136] and began to produce the Rapiscan Secure 1000. In 2007, the TSA signed a contract with Rapiscan to procure a customized version of the Secure 1000 for deployment in airport passenger screening [151].

We purchased a Rapiscan Secure 1000 from an eBay seller who had acquired it in 2012 at a surplus auction from a U.S. Government facility located in Europe [76]. The system was in unused condition. It came with operating and maintenance manuals as well as detailed schematics, which were a significant aid to reverse engineering. The system consists of two separate components: the scanner unit, a large enclosure that handles X-ray generation and detection under the control of a special purpose embedded system, and the user console, a freestanding cabinet that contains a PC with a keyboard and screen. The two components are connected by a 12 m cable.

The system we tested is a dual pose model, which means that the subject must turn around in order to be scanned from the front and back in two passes. TSA screening checkpoints used the Secure 1000 single pose model [111], which avoids this inconvenience by scanning from the front and back using a pair of scanner units. Our system was manufactured in about September 2006 and includes EPROM software version 2.1. Documents obtained under the Freedom of Information Act suggest that more recent versions of the hardware and software were used for airport screening [156, 145], and we highlight some of the known differences below. Consequently, we focus our analysis on fundamental weaknesses in the Secure 1000 design that we suspect also affect newer versions. A detailed analysis of TSA models might reveal additional vulnerabilities.

3.2.1 Backscatter Imaging

X-ray backscatter imaging exploits the unique properties of ionizing radiation to penetrate visual concealment and detect hidden contraband. The physical process which generates backscatter is Compton scattering, in which a photon interacts with a loosely bound or free electron and scatters in an unpredictable direction [33]. Other interactions, such as the photoelectric effect, are possible, and the fraction of photons that interact and which particular effect occurs depends on each photon's energy and the atomic composition of the mass. For a single-element material, the determining factor is its atomic number Z, while a compound material can be modeled by producing an "effective Z," or Z_{eff} [146].

Under constant-spectrum X-ray illumination, the backscattered intensity of a given point is largely determined by the atomic composition of matter at that location, and to a lesser extent its density. Thus, organic materials, like flesh, can be easily differentiated from materials such as steel or aluminum that are made from heavier elements.

The Secure 1000 harnesses these effects for contraband screening by operating as a "reverse camera," as illustrated in Figure 3.2. X-ray output from a centrally-located tube (operating at 50 kVp and 5 mA) passes through slits in shielding material: a fixed horizontal slit directly in front of a "chopper wheel," a rapidly spinning disk with four radial slits. This results in a narrow, collimated X-ray beam, repeatedly sweeping across the imaging field. During a scan, which takes about 5.7 s, the entire X-ray assembly moves vertically within the cabinet, such that the beam passes over every point of the scene in a series of scan lines.

As the beam sweeps across the scene, a set of 8 large X-ray detectors measures the intensity of the backscattered radiation at each point, by means of internal photomultiplier tubes (PMTs). The Secure 1000 combines the output of all 8 detectors, and sends the resulting image signal to the user console, which converts the time-varying signal into a 160×480 pixel monochrome image, with the intensity of each pixel determined by the Z_{eff} value of the surface of the scan subject represented by that pixel location.

3.2.2 Subsystems

Operator interface. The operator interacts with the Secure 1000 through the user console, a commodity x86 PC housed within a lockable metal cabinet. With our system, the user

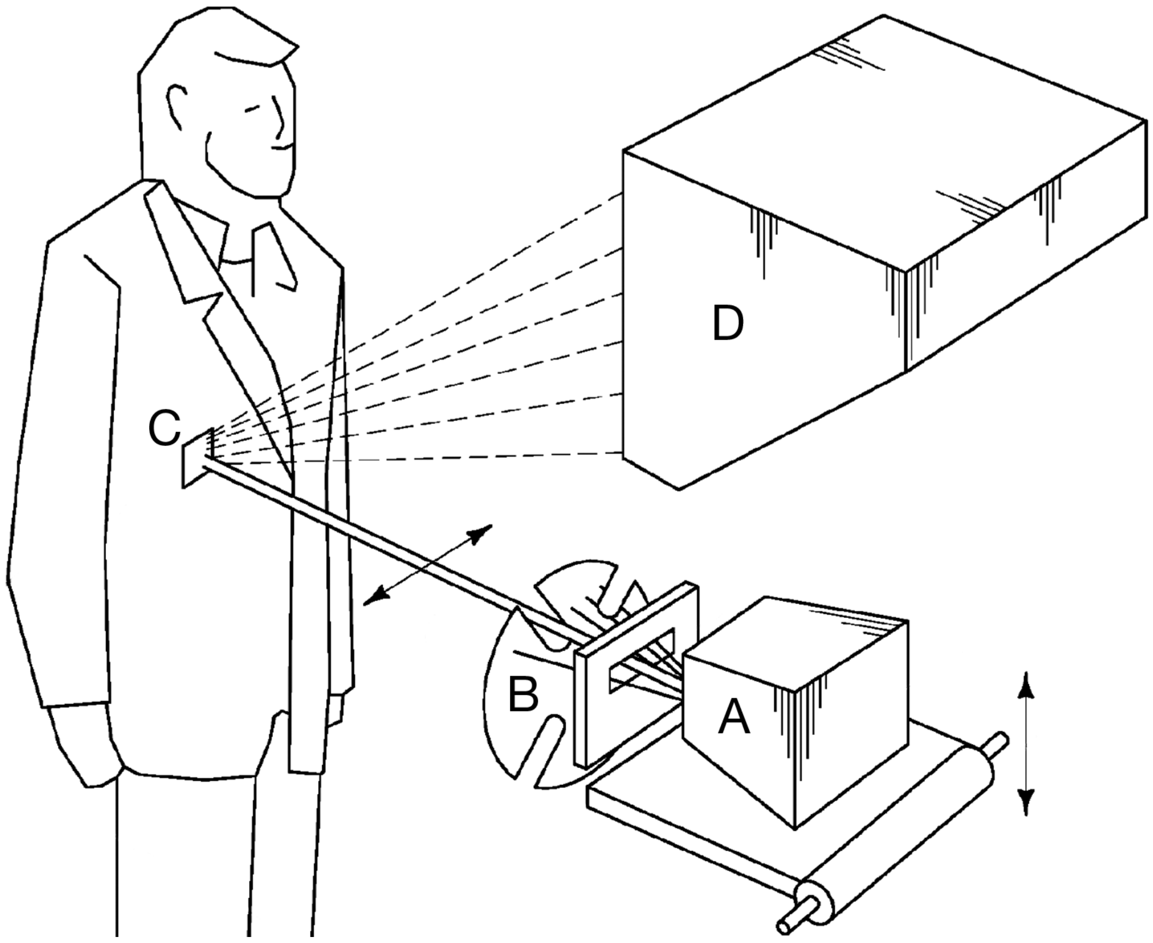


Figure 3.2: **Backscatter Imaging** — An X-ray tube (*A*) mounted on a platform travels vertically within the scanner. The X-rays pass through a spinning disk (*B*) that shapes them into a horizontally scanning beam. Some photons that strike the target (*C*) are backscattered toward detectors (*D*) that measure the reflected energy over time. Adapted from U.S. Patent 8,199,996 [72].

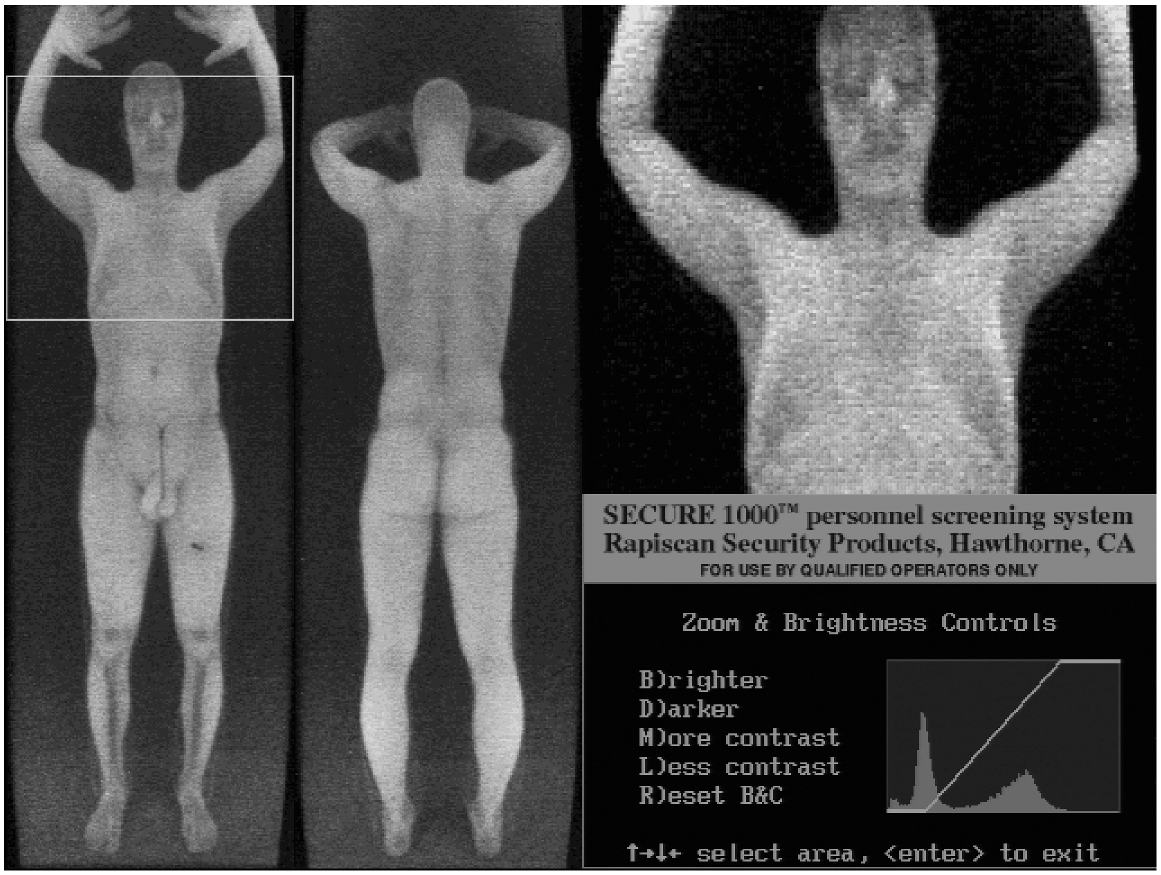

SECURE 1000™ personnel screening system
Rapiscan Security Products, Hawthorne, CA
FOR USE BY QUALIFIED OPERATORS ONLY
Zoom & Brightness Controls
B)righter
D)arker
M)ore contrast
L)ess contrast
R)eset B&C
↑→↓← select area, <enter> to exit

Figure 3.3: **Operator View** — The user console displays front and back images and offers basic enhancements and 2 × zoom. It also allows the operator to print images or save them to disk.

console is connected to the scanner unit via a serial link and an analog data cable. Documents released by the TSA indicate that airport checkpoint models were configured differently, with an embedded PC inside the scanner unit linked to a remote operator workstation via a dedicated Ethernet network [145, 156].

On our unit, the operator software is an MS-DOS application called SECURE65.EXE that launches automatically when the console boots. (TSA models are apparently Windows-based and use different operator software [151, 145].) This software is written in a BASIC variant, and the main user interface is a 640 × 480 pixel, 4-bit grayscale screen, as shown in Figure 3.3. The operator invokes a scan by pressing a hand switch. After image acquisition, the operator can inspect the scan by means of a 2× zoom and interactive brightness and contrast controls. The image can also be saved to disk or printed. Further, the software contains several calibration functions that can only be accessed by entering a 4 digit numeric password. The password is hard-coded and is printed in the maintenance manual.

Scanner unit. The scanner unit contains an assortment of electrical and mechanical systems under the control of an embedded computer called the System Control Board (SCB). The SCB houses an Intel N80C196KB12 microcontroller, executing software contained on a 32 KiB socketed ROM. It interacts with the user console PC over a bidirectional RS-232 serial link using simple ASCII commands such as `SU` for "scan up" and `SD` for "scan down." In turn, the SCB uses digital and analog interfaces to direct and monitor other components, including the X-ray tube, PMTs, and chopper wheel. It also implements hardware-based safety interlocks on the production of X-rays, which we discuss further in Section 3.4.2.

To control vertical movement of the X-ray tube, the scanner unit uses an off-the-shelf reprogrammable servo motor controller, the Parker Gemini GV6. In normal operation, the servo controller allows the SCB to trigger a movement of the X-ray tube, initially to a "home" position and subsequently to scan up and down at predefined rates. There is no command to move the tube to a specific intermediate position.

3.3 Contraband Detection

As the Secure 1000 is intended to detect prohibited or dangerous items concealed on the body of an attacker, the first and most obvious question to ask is how effectively the Secure 1000 detects contraband.

To make the discussion concrete, we consider the machine as it was typically used by the TSA for airport passenger screening. Under TSA procedures, subjects were imaged from the front and back, but not from the sides. A trained operator inspected the images and, if an anomaly was detected, the passenger was given a manual pat down to determine whether it was a threat [145]. The Secure 1000 was used in place of a walk-through metal detector, rather than both screening methods being employed sequentially [152]. We focus our analysis on threats relevant to an airport security context, such as weapons and explosives, as opposed to other contraband such as illicit drugs or bulk currency.

To replicate a realistic screening environment, we situated our Secure 1000 in an open area, oriented 2.5 m from a concrete wall sufficient to backstop X-ray radiation. This distance accords with the manufacturer's recommendation of at least 2 m of open area "for producing the best possible images" [121]. For typical tests, we arranged the subject at a distance of about 38 cm in front of the scanner using the foot position template provided with the machine.

Naïve adversary. First, we consider the scanner's effectiveness against a naïve adversary, an attacker whose tactics do not change in response to the introduction of the device.

Although this is a weak attacker, it seems to correspond to the threat model under which the scanner was first tested by the government, in a 1991 study of a prototype of the Secure 1000 conducted by Sandia National Laboratories [84]. Our results under this threat model generally comport with theirs. Guns, knives, and blocks of explosives naïvely carried on the front or back of the subject's body are visible to the scanner operator.

Three effects contribute to the detectability of contraband. The first is *contrast*: human skin appears white as it backscatters most incident X-ray radiation, while metals, ceramics, and bone absorb X-rays and so appear dark gray or black. The second is *shadows* cast by three-dimensional objects as they block the X-ray beam, which accentuate their edges. The third is *distortion* of the subject's flesh as a result of the weight of the contraband or the mechanics of its attachment. The naïve adversary is unlikely to avoid all three effects by chance.

A successful detection of hidden contraband can be seen in Figure 4.1. The subject has concealed a .380 ACP pistol within his waistband. The X-ray beam interacts with the gun metal significantly differently than the surrounding flesh, and the sharp contrast in backscatter intensity is immediately noticeable.

Adaptive adversary. Of course, real attackers are not entirely ignorant of the scanner. The TSA announced that it would be used at screening checkpoints [152, 50], the backscatter imaging mechanism is documented in patents and manufacturer reports [72, 88, 120], images captured with the device have appeared in the media [50, 89], and the physics of backscatter X-rays are well understood [33, 15, 84]. We must assume that attackers have such information and adapt their tactics in response.

To simulate an adaptive adversary, we performed experiments in the style of white-box penetration testing commonly employed in the computer security field. We allowed ourselves complete knowledge of how the scanner operates as well as the ability to perform test scans, observed the resulting images, and used them to adjust our concealment methods.

Such interactive testing is not strictly necessary to develop clever attacks. Indeed, researchers with no access to the Secure 1000 have proposed a number of concealment strategies based only on published information [83], and we experimentally confirm that several of these attacks are viable. However, the ability to perform tests substantially increases the probability that an attack will succeed on the first attempt against a real deployment. A determined adversary might acquire this level of access in several ways: by buying a machine, as we did; by colluding with a dishonest operator; or by probing the security of real installations over time.

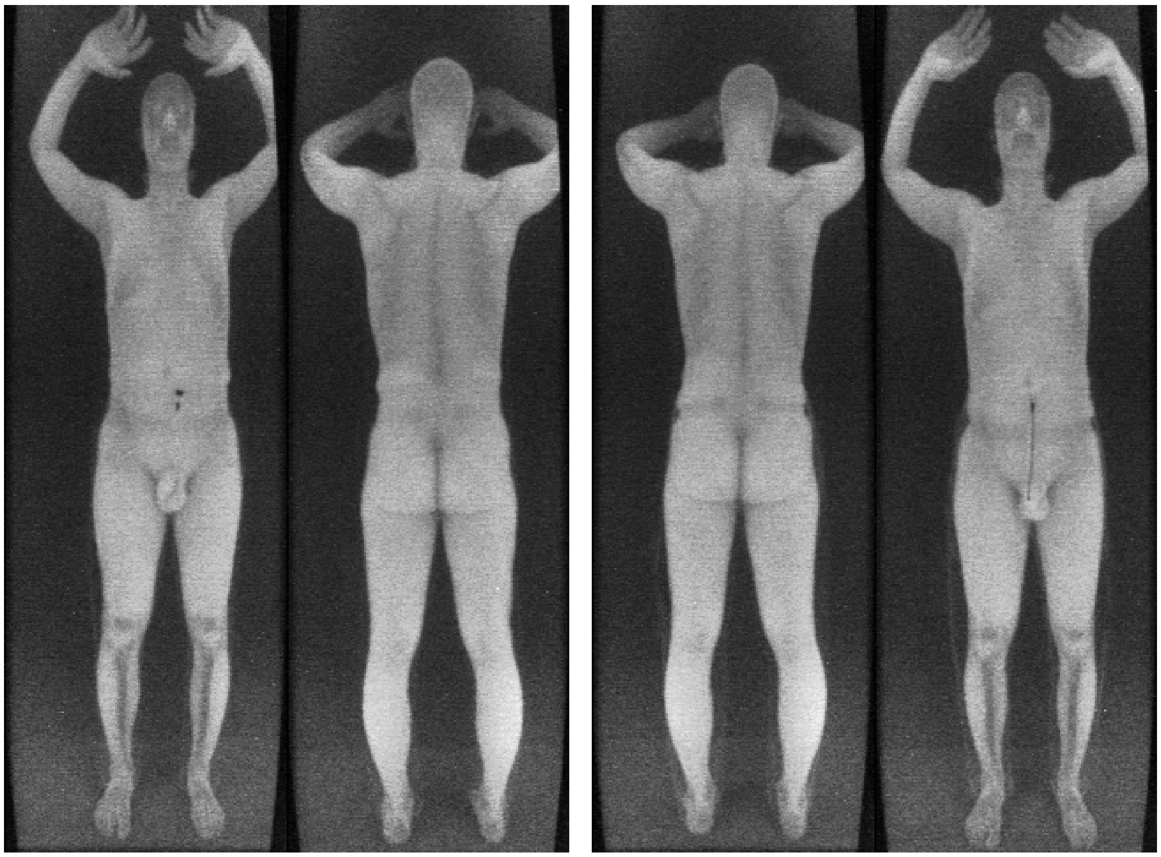


Figure 3.4: **Concealing a Pistol by Positioning** — The Secure 1000 cannot distinguish between high Z_{eff} materials, such as a metal handgun, and the absence of a backscatter response. Carefully placed metallic objects can be invisible against the dark background.

In the remainder of this section, we describe experiments with three adaptive concealment techniques and show that they can be used to defeat the Secure 1000. We successfully use them to smuggle firearms, knives, and explosive simulants past the scanner.

3.3.1 Concealment by Positioning

The first concealment technique makes use of a crucial observation about X-ray physics: backscatter screening machines emitting X-rays in the 50 keV range, such as the Secure 1000, cannot differentiate between the absence of matter and the existence of materials with high Z_{eff} (e.g., iron and lead). That is, when the scanner emits probing X-rays in a direction and receives no backscatter, it can either be because the beam interacted with nothing, i.e., traveled unimpeded past the screening subject, or because the beam shone directly upon a material which absorbed it entirely and thus did not backscatter. In either case, the resulting pixels will be dark.

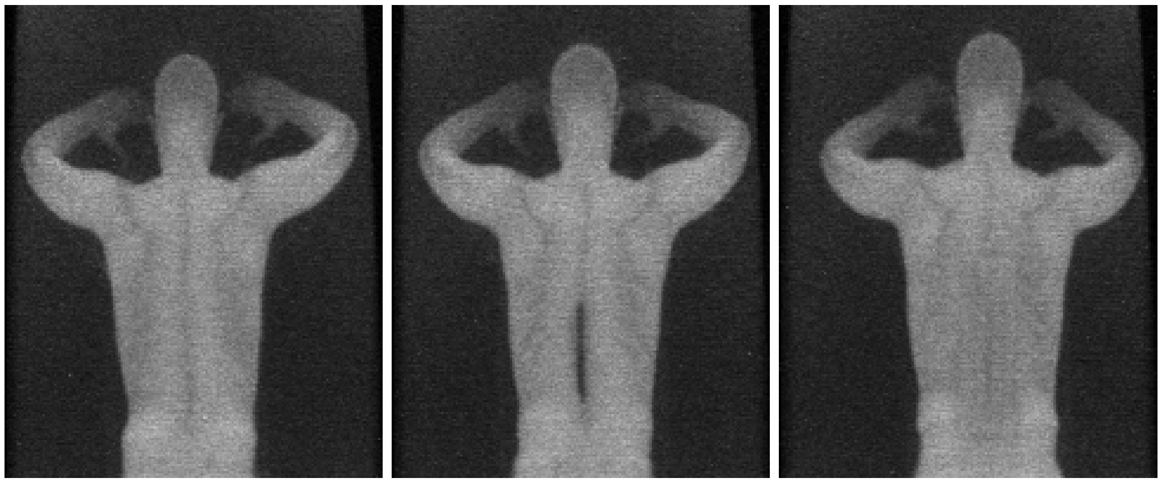


Figure 3.5: **Concealing a Knife by Masking**—We find that high-Z_{eff} materials can be hidden by covering them with lower Z_{eff} materials, such as the common plastic PTFE (Teflon). For example, a metal knife is clearly visible when naïvely concealed, but when covered with a thin plastic block it approximates the color of the spine. Tapering the block's edges would reduce the visible outline.

These facts lead directly to a straightforward concealment attack for high Z_{eff} contraband: position the object such that it avoids occluding the carrier's body with respect to the X-ray beam. This technique was first suggested on theoretical grounds by Kaufman and Carlson [83]. In limited trials, a TSA critic used it to smuggle small metal objects through airport checkpoints equipped with the Secure 1000 and other AITs [34]. Note that this attack is not enabled by a poor choice of image background color; as discussed above, the scanner cannot differentiate between the metal objects and the absence of material.

To more fully investigate this attack, we obtained a set of weapons: both knives and firearms, ranging from a .380 ACP pistol to an AR-15 semi-automatic rifle. When we scanned the weapons against a dark backdrop, most of the firearms were readily visible due to the presence of nonmetallic parts. After testing a number of firearms, we settled on our .380 ACP pistol as the most suitable candidate for concealment.

We performed several trials to test different placement and attachment strategies. In the end, we achieved excellent results with two approaches: carefully affixing the pistol to the outside of the leg just above the knee using tape, and sewing it inside the pant leg near the same location. Front and back scans for both methods are shown in Figure 3.4. In each case, the pistol is invisible against the dark background, and the attachment method leaves no other indication of the weapon's presence.

In a similar test, we concealed an 11 cm metal folding knife, in its closed position, along our test subject's side. In this case, too, front and back scans were completely unable to detect the weapon.

Fortunately, simple procedural changes can thwart these attacks. Instead of performing only front and back scans, every subject could also be made to undergo scans from the left and right sides. Under these scans, a high Z_{eff} weapon positioned on the side of the body would be as obvious as the one in Figure 4.1. Unfortunately, these additional scans would nearly halve the maximum throughput of the checkpoint, as well as double each person's radiation dose. Another possible mitigation would be to screen each subject with a magnetometer, which would unequivocally find metallic contraband but would fail to uncover more exotic weapons, such as ceramic knives [154, 158]. We note that the attacker's gait or appearance might be compromised by the mass and bulk of the firearm or knife, and this might be noticeable to security personnel outside of the backscatter X-ray screening.

3.3.2 Concealment by Masking

The second object concealment techniques we attempted are similarly based on X-ray physics: the brightness of a material in the image is directly correlated to its backscatter intensity, which in turn is determined by the Z_{eff} and density of the matter in the path of the beam. Therefore, any combination of substances which scatter incoming X-rays at the same approximate intensity as human flesh will be indistinguishable from the rest of the human.

One consequence of this fact is that high-Z_{eff} contraband can be concealed by masking it with an appropriate thickness of low-Z_{eff} material. We experimented with several masking materials to find one with a Z_{eff} value close to that of flesh. We obtained good results with the common plastic PTFE (Teflon), although due to its low density a significant thickness is required to completely mask a metallic object.

To work around this issue, we took advantage of the Secure 1000's ability to see bones close to the skin. Figure 3.5 demonstrates this approach: an 18 cm knife is affixed to the spine and covered with 1.5 cm of PTFE. As the X-rays penetrate through the material, they backscatter so that the knife outline approximates our subject's spine. While this mask arrangement creates hard edges and shadows which render it noticeable to screening personnel these effects could be reduced by tapering the edges of the mask.

A more difficult challenge for the attacker is taking into account the anatomy of the specific person being imaged. Shallow bones and other dense tissue are visible to the scanner under normal conditions, and a poorly configured mask will stand out against these darker areas of the scan. We conclude that masking can be an effective concealment technique, but achieving high confidence of success would require access to a scanner for testing.

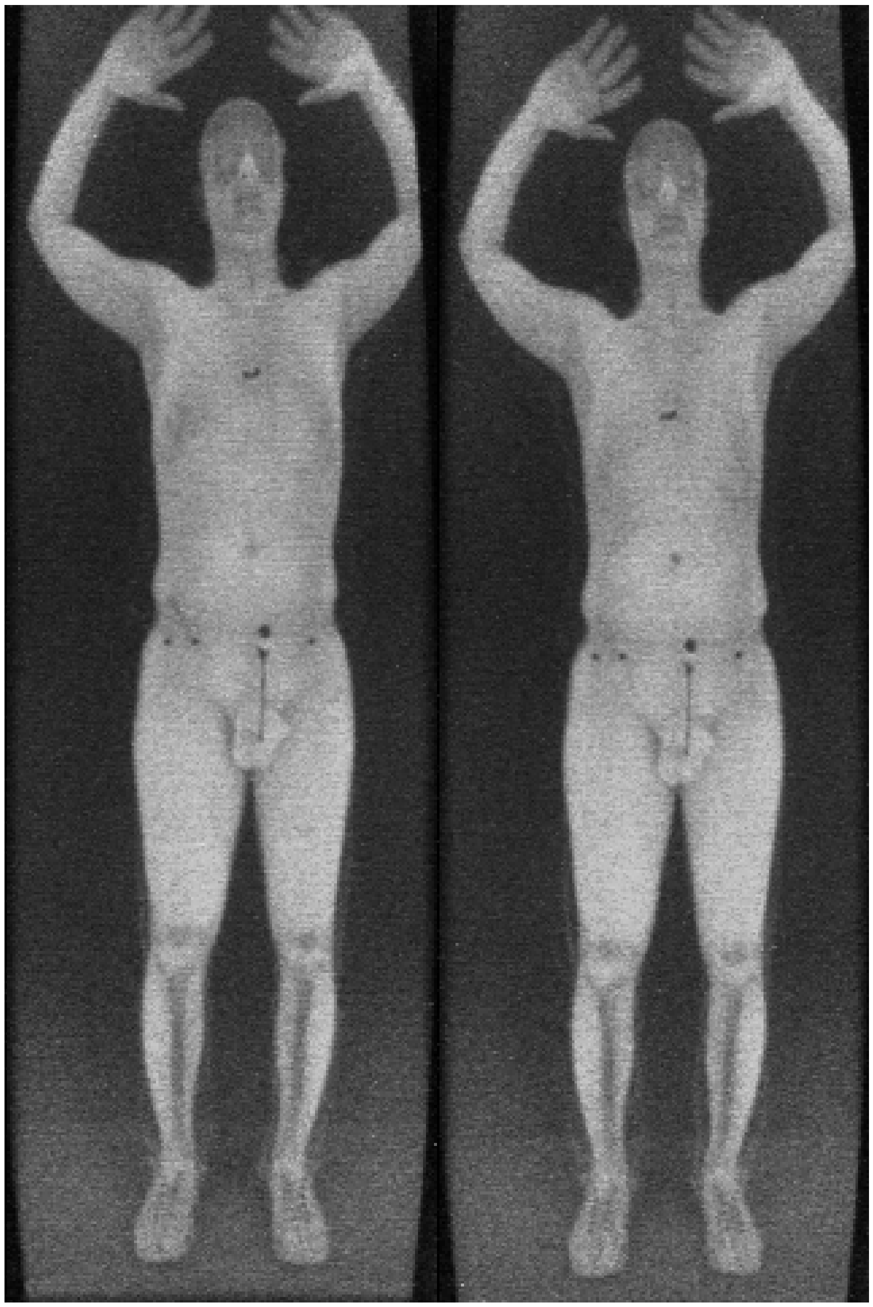


Figure 3.6: **Concealing Explosives by Shaping** — *Left:* Subject with no contraband. *Right:* Subject with more than 200 g of C-4 plastic explosive simulant plus detonator, molded to stomach.

3.3.3 Concealment by Shaping

Our third and final concealment technique applies a strategy first theorized in [83] to hide malleable, low-Z_{eff} contraband, such as plastic explosives. These materials produce low contrast against human flesh, and, unlike rigid weapons, the attacker can reshape them so that they match the contours of the body.

To experiment with this technique, we acquired radiological simulants for both Composition C-4 [176] and Semtex [177], two common plastic high explosives. These simulants are designed to emulate the plastic explosives with respect to X-ray interactions, and both are composed of moldable putty, similar to the actual explosive materials. We imaged both C-4 and Semtex simulants with the Secure 1000, and found that they appear very similar. We selected the C-4 simulant for subsequent tests.

Our initial plan was to modify the simulants' Z_{eff} to better match that of flesh, by thoroughly mixing in fine metallic powder. To our surprise, however, a thin pancake (about 1 cm) of unmodified C-4 simulant almost perfectly approximated the backscatter intensity of our subject's abdomen.

We affixed the pancake with tape (which is invisible to the Secure 1000), and faced two further problems. First, the pancake covered our subject's navel, which is normally clearly visible as a small black area in the scans. Second, by design, plastic explosives are almost completely inert without a matching detonator. These problems neatly solve each other: we attached a detonator, consisting of a small explosive charge in a metal shell, directly over our subject's navel. Since the detonator is coated in metal, it absorbs X-rays quite well and mimics the look of the navel in the final image.

Figure 3.6 shows a side-by-side comparison of our test subject both carrying no contraband and carrying 200 g of C-4 explosive and attached detonator. To put this amount in perspective, "Shoe Bomber" Richard Reid reportedly carried about 280 g of explosive material [32], and the bomb that destroyed Pan Am Flight 103 is thought to have contained 350 g of Semtex [162].

These scans indicate that plastic explosives can be smuggled through a Secure 1000 screening, since thin pancakes of these materials do not contrast strongly with flesh. While a metal detector would have been sufficient to detect the detonator we used, not all detonators have significant metal components.

In summary, an adaptive adversary can use several attack techniques to carry knives, guns, and plastic explosives past the Secure 1000. However, we also find that multiple iterations of experimentation and adjustment are likely necessary to achieve consistent success. The security of the Secure 1000, then, rests strongly on the adversary's inability to acquire access to the device for testing. However, since we were able to purchase a Secure 1000, it is reasonable to assume that determined attackers and well-financed terrorist groups can do so as well. We emphasize that procedural changes — specifically, performing side scans and supplementing the scanner with a magnetometer — would defeat some, though not all, of the demonstrated attacks.

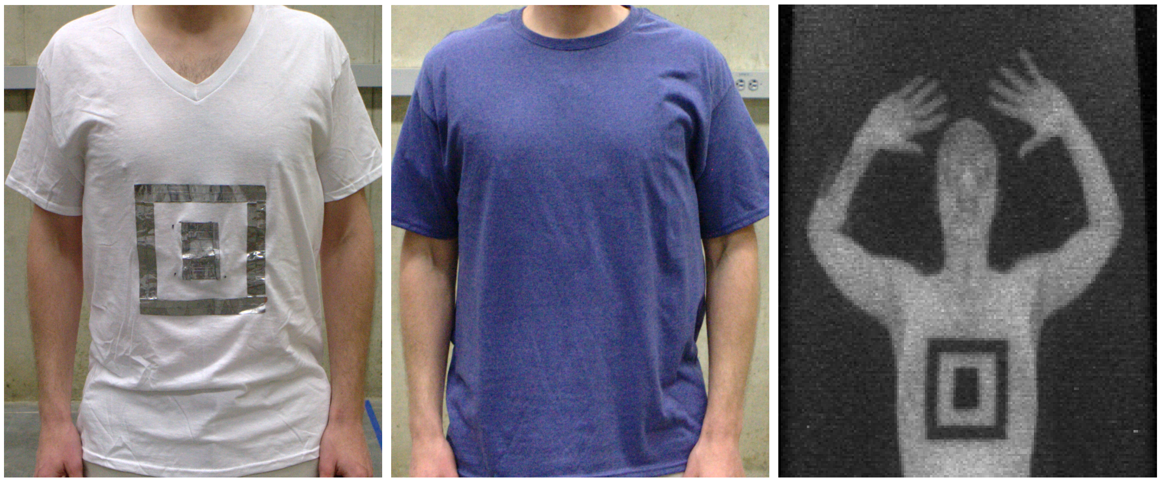


Figure 3.7: **A Secret Knock**—We demonstrate how malware infecting the Secure 1000 user console could be used to defeat the scanner. The malware is triggered when it detects a specific pattern in a scan, as shown here. It then replaces the real image (c) of the attacker, which might reveal hidden contraband, with an innocuous image stored on disk. Pattern recognition occurs in real time.

3.4 Cyberphysical Attacks

The Secure 1000, like other AITs, is a complex cyberphysical system. It ties together X-ray emitters, detectors, and analog circuitry under the control of embedded computer systems, and feeds the resulting image data to a traditional desktop system in the user console. In this section, we investigate computer security threats against AITs. We demonstrate a series of novel software- and hardware-based attacks that undermine the Secure 1000's efficacy, safety features, and privacy protections.

3.4.1 User Console Malware

The first threat we consider is malware infecting the user console. On our version of the Secure 1000, the user console is an MS-DOS–based PC attached to the scanner unit via a proprietary cable; TSA models apparently used Windows and a dedicated Ethernet switch [151, 153]. Although neither configuration is connected to an external network, there are several possible infection vectors. If the operators or maintenance personnel are malicious, they could abuse their access in order to manually install malware. The software on our machine lacks any sort of electronic access controls (e.g., passwords) or software verification. While the PC is mounted in a lockable cabinet, we were able to pick the lock in under 10 seconds with a commercially available tool. Therefore, even an outsider with temporary physical access could easily introduce malicious code. TSA systems may be better locked down, but sophisticated adversaries have a track record of infecting even highly secured, airgapped systems [92, 108].

We implemented a form of user console malware by reverse engineering SECURE65.EXE, the front-end software package used by the Secure 1000, and creating a malicious clone. Our version, INSECURE.EXE, is a functional, pixel-accurate reimplementation of the original program and required approximately one man-month to create.

In addition to enabling basic scanning operations, INSECURE.EXE has two malicious features. First, every scan image is saved to a hidden location on disk for later exfiltration. This is a straightforward attack, and it demonstrates one of many ways that software-based privacy protections can be bypassed. Of course, the user could also take a picture of the screen using a camera or smartphone — although operators are forbidden to have such devices in the screening room [127].

Second, INSECURE.EXE selectively subverts the scanner's ability to detect contraband. Before displaying each scan, it applies a pattern recognition algorithm to look for a "secret knock" from the attacker: the concentric squares of a QR code position block. If this pattern occurs, INSECURE.EXE replaces the real scan with a preprogrammed innocuous image. The actual scan, containing the trigger pattern and any other concealed contraband, is entirely hidden.

To trigger this malicious substitution, the subject simply wears the appropriate pattern, made out of any material with a sufficiently different Z_{eff} than human tissue. In our experiments, we arranged lead tape in the target shape, attached to an undershirt, as shown in Figure 3.7. When worn under other clothing, the target is easily detected by the malware but hidden from visual inspection.

Recently, in response to privacy concerns, the TSA has replaced manual review of images with algorithmic image analysis software known as automated target recognition (ATR) [155]. Instead of displaying an image of the subject, this software displays a stylized figure, with graphical indicators showing any regions which the software considers suspect and needing manual resolution. (Delays in implementing this algorithm led the TSA to remove Secure 1000 machines from airports entirely [12].) If malware can compromise the ATR software or its output path, it can simply suppress these indicators — no image replacement needed.

3.4.2 Embedded Controller Attacks

The System Control Board (SCB) managing the physical scanner is a second possible point of attack. While the SCB lacks direct control over scan images, it does control the scanner's mechanical systems and X-ray tube. We investigated whether an attacker who subverts the SCB firmware could cause the Secure 1000 to deliver an elevated radiation dose to the scan subject.

This attack is complicated by the fact that the Secure 1000 includes a variety of safety interlocks that prevent operation under unexpected conditions. Circuits sense removal of the front panel, continuous motion of the chopper wheel and the vertical displacement servo, X-ray tube temperature and supply voltage, X-ray production level, key position (“Standby” vs. “On”), and the duration of the scan, among other parameters. If any anomalous state is detected, power to the X-ray tube is immediately disabled, ceasing X-ray emission.

While some of these sensors merely provide inputs to the SCB software, others are tied to hard-wired watchdog circuits that cut off X-ray power without software mediation. However, the firmware can *bypass* these hardware interlocks. At the beginning of each scan, operational characteristics such as tube voltage and servo motion fluctuate outside their nominal ranges. To prevent immediate termination of every scan, SCB software temporarily asserts a bypass signal, which disables the hardware interlocks. This signal feeds a “bypass watchdog” circuit of its own, meant to prevent continual interlock bypass, but the SCB can pet this watchdog by continuously toggling the bypass signal, and cause all hardware interlocks to be ignored. Thus, every safety interlock is either directly under software control or can be bypassed by software.

We developed replacement SCB firmware capable of disabling all of the software and hardware safety interlocks in the Secure 1000. With the interlocks disabled, corrupt firmware can, for instance, move the X-ray tube to a specific height, stop the chopper wheel, and activate X-ray power, causing the machine to deliver the radiation dose from an entire dose to a single point. Only the horizontal displacement of this point is not directly under firmware control — it depends on where the chopper wheel happens to come to rest.

Delivering malicious SCB firmware presents an additional challenge. The firmware is stored on a replaceable socketed EPROM inside the scanner unit, which is secured by an easily picked wafer tumbler lock. Although attackers with physical access could swap out the chip, they could cause greater harm by, say, hiding a bomb inside the scanner. For SCB attacks to pose a realistic safety threat, they would need to be remotely deployable.

Due to the scanner’s modular design, the only feasible vector for remote code execution is the serial link between the user console and the SCB. We reverse engineered the SCB firmware and extensively searched for vulnerabilities. The firmware is simple (< 32 KiB) and appears to withstand attacks quite well. Input parsing uses a fixed length buffer, to which bytes are written from only one function. This function implements bounds checking correctly. Data in the buffer is always processed in place, rather than being copied to other locations that might result in memory corruption. We were unable to cause any of this code to malfunction in a vulnerable manner.

While we are unable to remotely exploit the SCB to deliver an elevated radiation dose, the margin of safety by which this attack fails is not reassuring. Hardware interlocks that can be bypassed from software represent a safety mechanism but not a security defense. Ultimately, the Secure 1000 is protected only by its modular, isolated design and by the simplicity of its firmware.

3.4.3 Privacy Side-Channel Attack

AIT screening raises significant privacy concerns because it creates a naked image of the subject. Scans can reveal sensitive information, including anatomical size and shape of body parts, location and quantity of fat, existence of medical conditions, and presence of medical devices such as ostomy pouches, implants, or prosthetics. As figures throughout the paper show, the resulting images are quite revealing.

Recognizing this issue, the TSA and scanner manufacturers have taken steps to limit access to raw scanned images. Rapiscan and DHS claim that the TSA machines had no capacity to save or store the images [97, 145]. The TSA also stated that the backscatter machines they used had a "privacy algorithm applied to blur the image" [154]. We are unable to verify these claims due to software differences between our machine and TSA models. Our Secure 1000 has documented save, recall (view saved images), and print features and does not appear to have a mechanism to disable them. In fact, using forensic analysis software on the user console's drive, we were able to recover a number of stored images from test scans that were incompletely deleted during manufacturing.

These software-based defenses aim to safeguard privacy in images that are constructed by the machine, but they do not address a second class of privacy attacks against AITs: an outsider observer could try to reconstruct scanned images by using their own external detector hardware. The most mechanically complex, dangerous, and energy intensive aspects of backscatter imaging are related to X-ray illumination; sensing the backscattered radiation is comparatively simple. Since X-rays scatter off the subject in a broad arc, they create a kind of physical side channel that potentially leaks a naked image of the subject to any nearby attacker. To the best of our knowledge, we are the first to propose such an attack; the privacy threat model for AITs appears to have been focused almost entirely on concerns about the behavior of screening personnel, rather than the general public.

In the scenario we envision, an attacker follows a target subject (for instance, a celebrity or politician) to a screening checkpoint while carrying an X-ray detector hidden in a

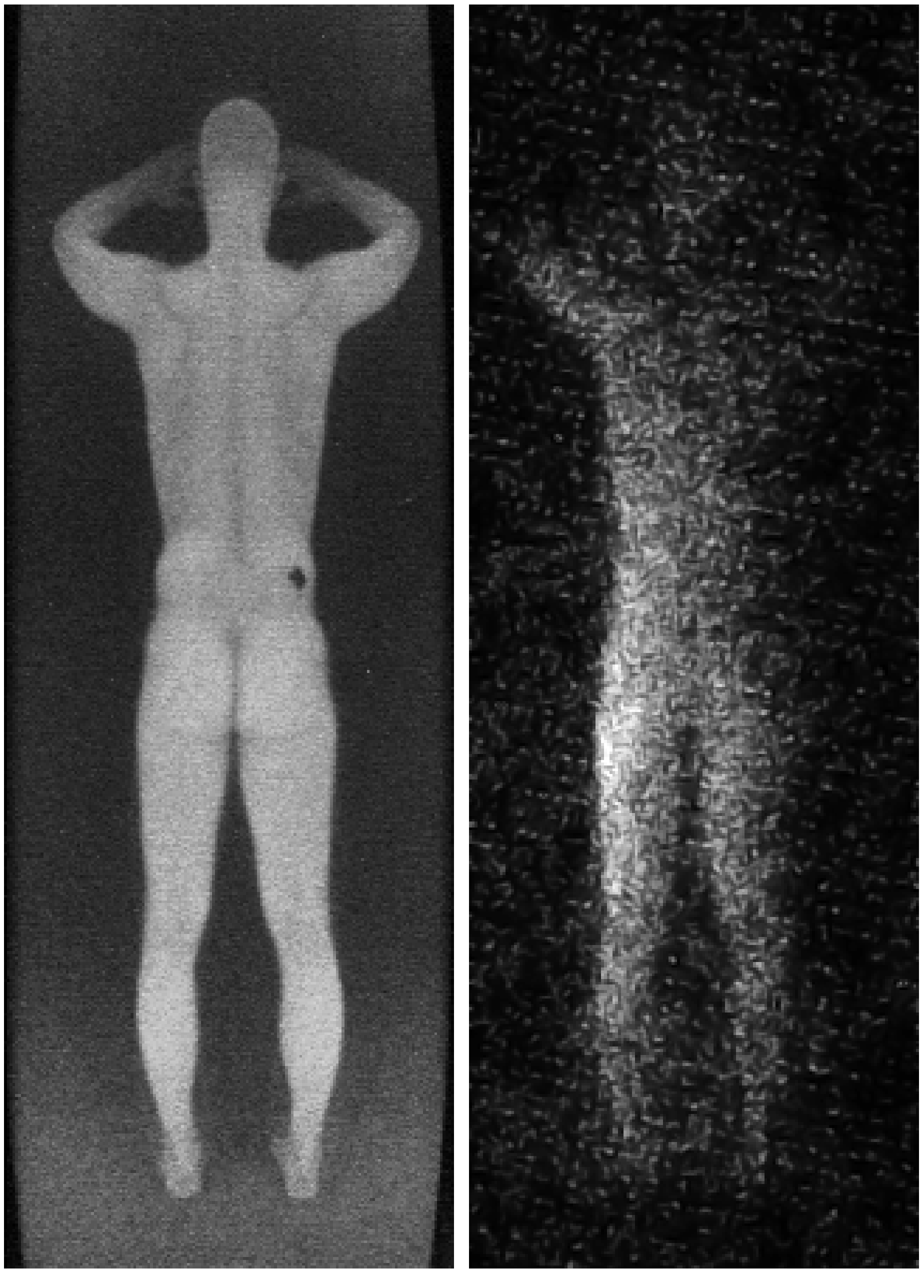


Figure 3.8: **Attacking Privacy** — An attacker could use a detector hidden in a suitcase to capture images of the subject during scanning. As a proof of concept, we used a small external PMT to capture images that are consistent with the scanner's output. A larger detector would produce more detailed images.

suitcase. As the victim is scanned, the hardware records the backscattered X-rays for later reconstruction.

We experimented with the Secure 1000 to develop a proof-of-concept of such an attack. The major technical challenge is gathering enough radiation to have an acceptable signal/noise ratio. The Secure 1000 uses eight large photomultiplier tubes (PMTs) — four on either side of the X-ray generator — in order to capture as much signal as possible. For best results, an attacker should likewise maximize observing PMT surface area, and minimize distance from the subject, as radiation intensity falls off quadratically with distance. To avoid arousing suspicion, an attacker may be limited to only one PMT, and may also be restricted in placement.

To determine whether external image reconstruction is feasible, we used a small PMT, a 75 mm Canberra model BIF2996-2 operated at 900 V, with a 10 cm × 10 cm NaI crystal scintillator. We placed this detector adjacent to the scanner and fed the signal to a Canberra Model 1510 amplifier connected to a Tektronix DPO 3014 oscilloscope. After capturing the resulting signal, we converted the time varying intensity to an image and applied manual enhancements to adjust levels and remove noise.

Figure 3.8 shows the results from the scanner and from our corresponding reconstruction. While our proof-of-concept results are significantly less detailed than the scanner's output, they suggest that a determined attacker, equipped with a suitcase-sized PMT, might achieve satisfactory quality. A further concern is that changes in future backscatter imaging devices might make this attack even more practical. Since the PMTs in the Secure 1000 are close to the maximum size that can fit in the available space, further improvements to the scanner's performance — i.e., better resolution or reduced time per scan — would likely require increased X-ray output. This would also increase the amount of information leaked to an external detector.

3.5 Discussion and Lessons

The Secure 1000 appears to perform largely as advertised in the non-adversarial setting. It readily detected a variety of naïvely concealed contraband materials. Our preliminary measurements of the radiation exposure delivered during normal scanning seem consistent with public statements by the manufacturer, TSA, and the FDA [158, 30, 78, 123]. Moreover, it seems clear that the manufacturer took significant care to ensure that predictable equipment malfunctions would not result in unsafe radiation doses; in order for this to happen a number of independent failures would be required, including failures of safety interlocks specifically designed to prevent unsafe conditions.

However, the Secure 1000 performs less well against clever and adaptive adversaries, who can use a number of techniques to bypass its detection capabilities and to attempt to subvert it by cyberphysical means. In this section, we use the device's strengths and weaknesses to draw lessons that may help improve the security of other AITs and cyberphysical security systems more generally.

The effectiveness of the device is constrained by facts of X-ray physics . . . As discussed in Section 3.2.1, Compton scattering is the physical phenomenon which enables backscatter imaging. As the tight beam of X-rays shines upon the scene, it interacts with the scene material. The intensity and energy spectrum of the backscattered radiation is a function of both the X-ray spectrum emitted by the imaging device and the atomic composition of the material in the scene.

The Secure 1000 emits a single constant X-ray spectrum, with a maximum energy of 50 keV, and detects the intensity of backscatter to produce its image. Any two materials, no matter their actual atomic composition, that backscatter the same approximate intensity of X-rays will appear the same under this technology. This physical process enables our results in Section 3.3.3. This issue extends beyond the Secure 1000: any backscatter imaging device based upon single-spectrum X-ray emission and detection will be vulnerable to such attacks.

By contrast, baggage screening devices (such as the recently studied Rapiscan 522B; see [122]) usually use transmissive, rather than backscatter, X-ray imaging. These devices also often apply dual-energy X-ray techniques that combine information from low-energy and high-energy scans into a single image. To avoid detection by such systems, contraband will need to resemble benign material under two spectra, a much harder proposition.

. . . but physics is irrelevant in the presence of software compromise. In the Secure 1000, as in other cyberphysical screening systems, the image of the object scanned is processed by software. If that software has been tampered with, it can modify the actual scan in arbitrary ways, faking or concealing threats. Indeed, the ability of device software to detect threats and bring them to the attention of the operator is presumed in the "Automated Target Recognition" software used in current TSA millimeter-wave scanners [155]. Automatic suppression of threats by malicious software is simply the (easier to implement) dual of automatic threat detection. As we show in Section 3.4.1, malware can be stealthy, activating only when it observes a "secret knock."

Software security, including firmware updates, networked access, and chain-of-custody for any physical media, must be considered in any cyberphysical scanning system. Even so, no publicly known study commissioned by TSA considers software security.

Procedures are critical, but procedural best practices are more easily lost than those embedded in software. As early as 1991, Sandia National Labs recommended the use of side scans to find some contraband:

> A metallic object on the side of a person would blend in with the background and be unobserved. However, a side scan would provide an image of the object.

> There are other means of addressing this which IRT is considering presently [84, page 14].

Yet TSA procedures appear to call for only front and back scans, and the device manual characterizes side scans as an unusual practice:

> The Secure 1000 can conduct scans in four subject positions, front, rear, left side and right side. Most users only conduct front and rear scans in routine operations and reserve the side scans for special circumstances [121, page 3-7].

Omitting side scans makes it possible to conceal firearms, as we discuss in Section 3.3.1.

Since side scans are necessary for good security, the device's design should encourage their use by default. Yet, if anything, the scanner user interface nudges operators away from performing side scans. It allows the display of only two images at a time, making it poorly suited to taking four scans of a subject. A better design would either scan from all sides automatically (the Secure 1000 is already sold in a configuration that scans from two sides without the subject's turning around) or encourage/require a four-angle scan.

Adversarial thinking, as usual, is crucial for security. The Sandia report concludes that both C-4 and Detasheet plastic explosives are detected by the Secure 1000. Attached to their report is an image from one C-4 test (Figure 3.9), wherein a 0.95 cm thick C-4 block is noticeable only by edge effects — it is outlined by its own shadow, while the intensity within the block almost exactly matches the surrounding flesh. This suggests a failure to think adversarially: since plastic explosives are, by design, moldable putty, the attacker can simply gradually thin and taper the edges of the mass, drastically reducing edge effects and rendering it much less noticeable under X-ray backscatter imaging. We describe precisely such an attack in Section 3.3.3.

The basic problem appears to be that the system, while well engineered, appears not to have been designed, documented, or deployed with adaptive attack in mind. For instance, attaching contraband to the side of the body as described in Section 3.3.1 is a straightforward attack that is enabled by scanning only straight-on rather than from all angles. However, the operator's manual shows only example images where the contraband is clearly at the front or the back.

The other attacks we describe in Sections 3.3 and 3.4, which allow us to circumvent or weaken the advertised efficacy, privacy, and security claims, again show that the system's designers failed to think adversarially.

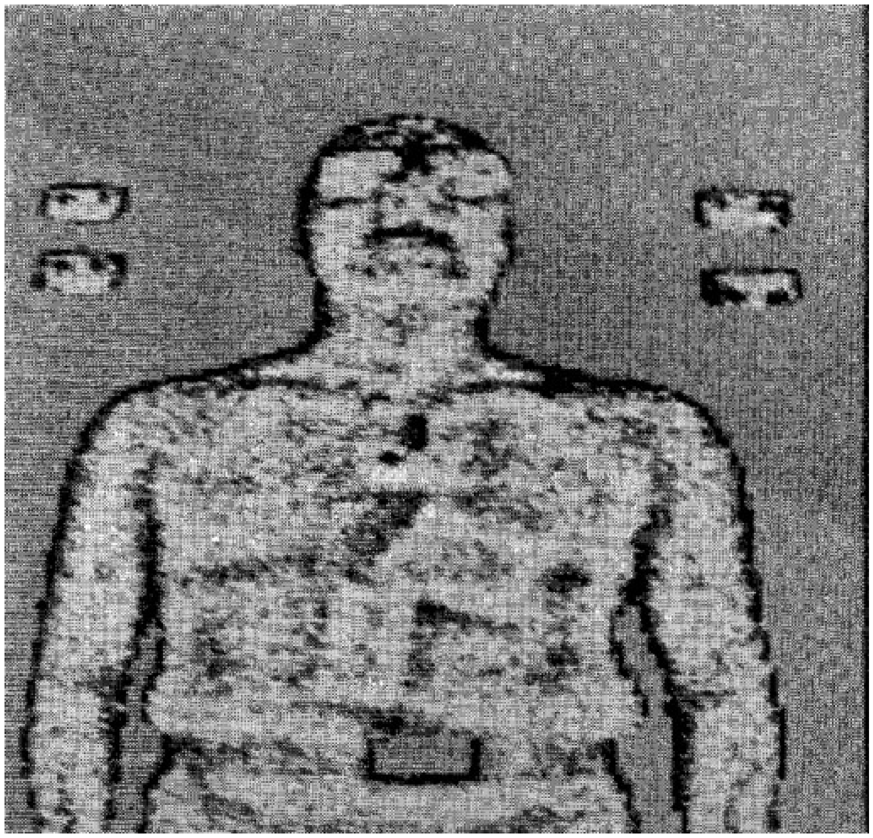


Figure 3.9: **Naïve Evaluation** — In an evaluation by Sandia National Labs, a Secure 1000 prototype successfully detects blocks of C-4 plastic explosive and Lucite attached to the subject's chest. Observe that the detection is based almost entirely on the X-ray shadow surrounding each rectangular block, which can be reduced or eliminated by an adaptive adversary through clever shaping and positioning of contraband. Reproduced from [84].

Simplicity and modular design are also crucial for security. The system control board implements simple, well-defined functionality and communicates with the operator console by means of a simple protocol. We were unable to compromise the control board by abusing the communication protocol. This is in contrast to the scanner console, whose software runs on a general-purpose COTS operating system.

Simplicity and modular design prevented worse attacks, but do other AITs reflect these design principles? Modern embedded systems tend towards greater integration, increased software control, and remote network capabilities, which are anathema to security.

Components should be designed with separation of concerns in mind: each component should be responsible for controlling one aspect of the machine's operation. Communication between components should be constrained to narrow data interfaces. The Secure 1000 gets these principles right in many respects. For example, the PC software does not have the ability to command the X-ray tube to a particular height. Instead, it can only command the tube to return to its start position or to take a scan.

Our main suggestion for improving the Secure 1000's cyberphysical security is to remove the ability for the control board firmware to override the safety interlocks (something currently needed only briefly, at scan initialization). As long as this bypass functionality is in place, the interlocks can serve as safety mechanisms but not as a defense against software- or firmware-based attacks.

Keeping details of the machine's behavior secret didn't help . . . Published reports about the Secure 1000 have been heavily redacted, omitting even basic details about the machine's operation. This did not stop members of the public from speculating about ways to circumvent the machine, using only open-source information. In an incident widely reported in the press, Jonathan Corbett suggested that firearms hanging off the body might be invisible against the dark background [34], an attack we confirm and refine in Section 3.3.1. Two physicists, Leon Kaufman and Joseph Carlson, reverse engineered the Secure 1000's characteristics from published scans and concluded that "[i]t is very likely that a large (15–20 cm in diameter), irregularly-shaped, [one] cm-thick pancake [of plastic explosive] with beveled edges, taped to the abdomen, would be invisible to this technology" [83], an attack we confirm and refine in Section 3.3.3. Keeping basic information about the device secret made an informed public debate about its use at airports more difficult, but did not prevent dangerous attacks from being devised.

. . . but keeping attackers from testing attacks on the machine might. To a degree that surprised us, our attacks benefited from testing on the device itself. Our first attempts at implementing a new attack strategy were often visible to the scanner, and reliable concealment was made possible only by iteration and refinement. It goes without saying that software-replacement attacks on the console are practical only if one has a machine to reverse engineer. As a result, we conclude that, in the case of the Secure 1000, keeping the machine out of the hands of would-be attackers may well be an effective strategy for preventing reliable exploitation, even if the details of the machine's operation were disclosed.

The effectiveness of such a strategy depends critically on the difficulty of obtaining access to the machine. In addition to the device we purchased, at least one other Secure 1000 was available for sale on eBay for months after we obtained ours. We do not know whether it sold, or to whom. Also, front-line security personnel will always have some level of access to the device at each deployment installation (including at non-TSA facilities) as they are responsible for its continued operation. Given these facts, imposing stricter purchase controls on backscatter X-ray machines than those currently enacted may not be enough to keep determined adversaries from accessing, studying, and experimenting with them.

3.6 Related work

Cyberphysical devices must be evaluated not only for their safety but also for their security in the presence of an adversary [79]. This consideration is especially important for AITs, which are deployed to security checkpoints. Unfortunately, AIT manufacturers and TSA have not, to date, allowed an unfettered independent assessment of AITs. Security evaluators retained by a manufacturer or its customers may not have an incentive to find problems [106]. In the case of a backscatter X-ray AIT specifically, an evaluation team may be skilled in physics but lack the expertise to identify software vulnerabilities, or vice versa.

Ours is the first study to consider computer security aspects of an AIT's design and operation, and the first truly independent assessment of an AIT's security, privacy, and efficacy implications informed by experimentation with an AIT device.

Efficacy and procedures. In 1991, soon after its initial development, the Secure 1000 was evaluated by Sandia National Laboratories on behalf of IRT Corp., the company then working to commercialize the device. The Sandia report [84] assessed the device's effectiveness in screening for firearms, explosives, nuclear materials, and drugs. The Sandia evaluators do not appear to have considered adaptive strategies for positioning and shaping contraband, nor did they consider attacks on the device's software. Nevertheless, they observed that side scans were sometimes necessary to detect firearms.

More recently, the Department of Homeland Security's Office of Inspector General released a report reviewing TSA's use of the Secure 1000 [37]. This report proposed improvements in TSA procedures surrounding the machines but again did not consider adversarial conditions or software vulnerabilities.

Working only from published descriptions of the device, researchers have hypothesized that firearms can be concealed hanging off the body [34] and that plastic explosives can be caked on the body [83]. We confirm these attacks are possible in Section 3.3 and refine them through access to the device for testing.

Health concerns. The ionizing radiation used by the Secure 1000 poses at least potential health risks. Studies performed on behalf of TSA by the Food and Drug Administration's Center for Devices and Radiological Health [30] and by the Johns Hopkins University Applied Physics Laboratory [78] attempted to quantify the overall radiation dose delivered by the device. Both studies saw public release only in heavily redacted form, going so far as to redact even the effective current of the X-ray tube.

In 2010, Professors at the University of California, San Francisco wrote an open letter to John P. Holdren, the Assistant to the President for Science and Technology, expressing their concern about potential health effects from the use of backscatter X-ray scanners at airports [131]. The letter writers drew on their radiological expertise, but did not have access to a Secure 1000 to study. The FDA published a response disputing the technical claims in the UCSF letter [101], as did the inventor of the Secure 1000, Steven W. Smith [136]. Under dispute was not just the total radiation dose but its distribution through the skin and body. In independent work concurrent with ours, a task group of the American Association of Physicists in Medicine [15] explicitly considered skin dose. The task group's measurements are within an order of magnitude of our own.

3.7 Conclusion

We obtained a Rapiscan Secure 1000 and evaluated its effectiveness for people screening. Ours was the first analysis of an AIT that is independent of the device's manufacturer and its customers; the first to assume an adaptive adversary; and the first to consider software as well as hardware. By exploiting properties of the Secure 1000's backscatter X-ray technology, we were able to conceal knives, firearms, plastic explosive simulants, and detonators. We further demonstrated that malicious software running on the scanner console can manipulate rendered images to conceal contraband.

Our findings suggest that the Secure 1000 is ineffective as a contraband screening solution against an adaptive adversary who has access to a device to study and to use for testing and refining attacks. The flaws we identified could be partly remediated through changes to procedures: performing side scans in addition to front and back scans, screening subjects with magnetometers as well as backscatter scanners, or including human pat-downs; but these procedural changes will lengthen screening times and increase cost.

One root cause of many of the issues we describe seems to be a failure of the system designers to think adversarially. That failure extends also to publicly available evaluations of the Secure 1000's effectiveness. Additionally, the secrecy surrounding AITs has sharply limited the ability of policymakers, experts, and the general public to assess the government's safety and security claims.

Ultimately, this study illustrates the risks of implementing checks in hardware and software. By creating a deterministic process for detecting contraband, clever adversaries can develop tricks and procedures that are all but guaranteed to subvert these checks. In this case, a machine blindly compares the amount of backscatter off of a material to determine if it is contraband or skin. This specifically has two problems: first, that the machine does not "understand" or "know" that it should be looking for contraband, and second, the proxy used for detecting contraband (amount of X-ray backscatter) is a poor one, as skin and contraband can be made to look similar under adversarial conditions.

Despite these flaws, other defenses may still make it hard for attackers to compromise the security of airplanes. Hardened cockpit doors may mitigate the hijacking threat from firearms and knives, but what is clearly needed, with or without AITs, is a robust means for detecting explosives. The millimeter-wave scanners currently deployed to airports will likely behave differently from the backscatter scanner we studied. We recommend that those scanners, as well as any future AITs — whether of the millimeter-wave or backscatter [115] variety — be subjected to independent, adversarial testing as a minimum bar to deployment.

CHAPTER 4

Internet Censorship

4.1 Introduction

In recent years, governments have looked toward applying strict censorship policies to their citizens Internet traffic. Governments such as China and Iran have focused on restricting what websites and services can be accessed within their respective countries [49, 148, 19]. Typically, these censors operate by inspecting packets at network routers at the edge of the country's network [178]. These routers are programmed to carry out the censors' *intent* to block access to certain information. However, because these routers can not understand this intent, they may allow requests or responses through that the censor wishes to block. One example is users who employ proxies to circumvent the government's imperfect censorship system. Here, the citizens are the adversaries who wish to exploit bugs and shortcomings in the government's censorship in order to access websites freely. In this chapter, we will investigate a novel way that citizens and activists can use to circumvent deployed censorship.

Today, the most widely-used tools for circumventing Internet censorship take the form of encrypted tunnels and proxies, such as Dynaweb [48], Instasurf [157], and Tor [42]. While these designs can be quite effective at sneaking client connections past the censor, these systems inevitably lead to a cat-and-mouse game in which the censor attempts to discover and block the services' IP addresses. For example, Tor has recently observed the blocking of entry nodes and directory servers in China and Iran [148]. Though Tor is used to skirt Internet censors in these countries, it was not originally designed for that application. While it may certainly achieve its original goal of anonymity for its users, it appears that Tor and proxies like it are ultimately not enough to circumvent aggressive censorship.

To overcome this problem, we propose *Telex*: an "end-to-middle" proxy with no IP address, located within the network infrastructure. Clients invoke the proxy by using public-key steganography to "tag" otherwise ordinary TLS sessions destined for uncensored websites. Its design is unique in several respects:

Architecture Previous designs have assumed that anticensorship services would be provided by hosts at the edge of the network, as the end-to-end principle requires. We propose instead to provide these services in the core infrastructure of the Internet, along paths between the censor's network and popular, nonblocked destinations. We argue that this will provide both lower latency and increased resistance to blocking.

Deployment Many systems attempt to combat state-level censorship using resources provided primarily by volunteers. Instead, we investigate a government-scale response based on the view that state-level censorship needs to be combated by state-level anticensorship.

Construction We show how a technique that the security and privacy literature most frequently associates with government surveillance—deep-packet inspection—can provide the foundation for a robust anticensorship system.

We expect that these design choices will be somewhat controversial, and we hope that they will lead to discussion about the future development of anticensorship systems.

Contributions and roadmap We propose using "end-to-middle" proxies built into the Internet's network infrastructure as a novel approach to resisting state-level censorship. We elaborate on this concept and sketch the design of our system in Section 4.2.

We develop a new steganographic tagging scheme based on elliptic curve cryptography, and we use it to construct a modified version of the TLS protocol that allows clients to connect to our proxy. We describe the tagging scheme in Section 4.4 and the protocol in Section 4.5. We analyze the protocol's security in Section 4.6. Finally, we present a proof-of-concept implementation of our approach and protocols in Section 4.7.

The material in this chapter is adapted from "Telex: Anticensorship in the Network Infrastructure" by Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman, which originally appeared in *Proceedings of the 20th USENIX Security Symposium*, August 2011.

4.2 Concept

Telex operates as what we term an "end-to-middle" proxy. Whereas in traditional end-to-end proxying the client connects to a server that relays data to a specified host, in end-to-middle proxying an intermediary along the path to a server redirects part of the connection payload to an alternative destination. One example of this mode of operation is Tor's leaky-pipe circuit topology [42] feature, which allows traffic to exit from the middle of a constructed Tor circuit rather than the end.

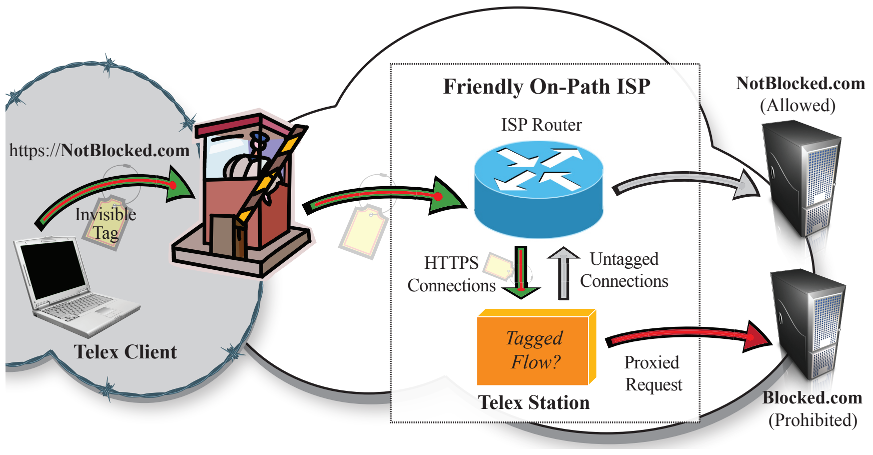


Figure 4.1: **Telex Concept** — This figure shows an example user connecting to Telex. The client makes a tagged connection to NotBlocked.com, which is passed by the censor's filter. When the request reaches a friendly on-path ISP, one of the ISP's routers forwards the request to the Telex station connected to its tap interface. Telex deciphers the tag, instructs the router to block the connection to NotBlocked.com and diverts the connection to Blocked.com, as the user secretly requested. If the connection was not tagged, Telex would not intervene, and it would proceed to NotBlocked.com as normal.

The Telex concept is to build end-to-middle proxying capabilities into the Internet's routing infrastructure. This would let clients invoke proxying by establishing connections to normal, pre-existing servers. By applying this idea to a widely used encrypted transport, such as TLS, and carefully avoiding observable deviations from the behavior of nonproxied connections, we can construct a service that allows users to robustly bypass network-level censorship without being detected.

In the remainder of this section, we define a threat model and goals for the Telex system. We then give a sketch of the design and discuss several practical considerations.

4.2.1 Threat model

Our adversary, "the censor", is a repressive state-level authority that desires to inhibit online access to information and communication of certain ideas. These desires are realized by IP and DNS blacklists as well as heuristics for blocking connections based on their observed content.

We note that the censor still has some motivation for connecting to the Internet at all, such as the economic and social benefits of connectivity. Thus, the censor bears some cost from over-blocking. We assume that the censor follows a blacklist approach rather than a whitelist approach in blocking, allowing traffic to pass through unchanged unless it is explicitly banned.

Furthermore, we assume that the censor generally permits widespread cryptographic protocols, such as TLS, except when it has reason to believe a particular connection is being used for skirting censorship. We further assume that the censor is not subverting such protocols on a wide scale, such as by requiring a cryptographic backdoor or by issuing false TLS certificates using a country-wide CA. We believe this is reasonable, as blocking or subverting TLS on a wide scale would render most modern websites unusably insecure. Subversion in particular would result in an increased risk of large-scale fraud if the back door were compromised or abused by corrupt insiders.

The censor controls the infrastructure of the network within its jurisdiction ("the censor's network"), and it can potentially monitor, block, alter, and inject traffic anywhere within this region. However, these abilities are subject to realistic technical, economic, and political constraints.

In general, the censor does *not* control end hosts within its network, which operate under the direction of their users. We believe this assumption is reasonable based on the failure of recent attempts by national governments to mandate client-side filtering software, such as China's Green Dam Youth Escort [172]. The censor might target a small subset of users and seize control of their devices, either through overt compulsion or covert technical attacks. Protecting these users is beyond the scope of our system. However, the censor's targeting users on a wide scale might have unacceptable political costs.

The censor has very limited abilities outside its network. It does not control any external network infrastructure or any popular external websites the client may use when communicating with Telex stations. The censor can, of course, buy or rent hosting outside its network, but its use is largely subject to the policies of the provider and jurisdiction.

Some governments may choose to deny their citizens Internet connectivity altogether, or disconnect entirely in times of crisis. These are outside our threat model; the best approaches to censors like these likely involve different approaches than ours, and entail much steeper performance trade-offs. Instead, our goal is to make access to any part of the global Internet sufficient to access every part of it. In other words, we aim to make connecting to the global Internet an all-or-nothing proposition for national governments.

4.2.2 Goals

Telex should satisfy the following properties:

Unblockable The censor should not be able to deny service to Telex without incurring unacceptable costs. In particular, we require that the censor cannot block Telex without blocking a large, primarily legitimate category of Internet traffic.

Confidential The censor should not be able to determine whether a user is using Telex or what content the user is accessing through the system.

Easy to deploy The consequences of Telex system failure (or even normal operation) must not interfere with normal network operation (e.g., non-Telex connections) in order for deployment to be palatable to ISPs.

Transparent to users Using Telex should, possibly after a small startup procedure, closely resemble using an unfiltered Internet connection.

4.2.3 Design

To meet our goals and the constraints imposed by our threat model, we propose the design shown in Figure 4.1. As illustrated in the figure, a Telex connection proceeds as follows:

1. The user's client selects an appropriate website that is not on the censor's blacklist and unlikely to attract attention, which we represent by the domain NotBlocked.com.
2. The user connects to NotBlocked.com via HTTPS. Her Telex client[1] includes an invisible "tag," which looks like an expected random nonce to the censor, but can be cryptographically verified by the Telex station using its private key.
3. Somewhere along the route between the client and NotBlocked.com, the connection traverses an ISP that has agreed to attach a Telex station to one of its routers. The connection is forwarded to the station via a dedicated tap interface.
4. The station detects the tag and instructs the router to block the connection from passing through it, while still forwarding packets to the station through its dedicated tap. (Unlike a deployment based on transparent proxying, this configuration *fails open*: it tolerates the failure of the entire Telex system and so meets our goal of being easy to deploy.)
5. The Telex station diverts the flow to Blocked.com as the user requested; it continues to actively forward packets from the client to Blocked.com and vice versa until one side terminates the connection. If the connection was untagged, it would pass through the ISP's router as normal.

[1]We anticipate that client software will be distributed out of band, perhaps by sneakernet, among mutually trusting individuals within the censor's domain.

We simplified the discussion above in an important point: we need to specify what protocol is to be used over the encrypted tunnel between the Telex client and the Telex station and how the client communicates its choice of Blocked.com. Layering IP atop the tunnel might seem to be a natural choice, yielding a country-wide VPN of sorts, but even a passive attacker may be able to differentiate VPN traffic patterns from those of a normal HTTPS connection. As a result, we primarily envision using Telex for protocols whose session behavior resembles that of HTTPS. For example, an HTTP or SOCKS proxy would be a useful application, or perhaps even an entry node (or list of entry nodes) for another anticensorship system such as Tor [42]. In the remainder of this chapter, we assume that the application is an HTTP proxy.

The precise placement of Telex stations is a second issue. Clearly, a chief objective of deployment is to cover as many paths between the censor and popular Internet destinations as possible so as to provide a large selection of sites to play the role of NotBlocked.com. We might accomplish this either by surrounding the censor with Telex stations or by placing them close to clusters of popular uncensored destinations. In the latter case, care should be taken not to reduce the size of the cluster such that the censor would only need to block a small number of otherwise desirable sites to render the station useless. Which precise method of deployment would be most effective and efficient is, in part, a geopolitical question.

A problem faced by existing anticensorship systems is providing sufficient incentives for deployment [35]. Whereas systems that require cooperation of uncensored websites create a risk that such sites might be blocked by censors in retaliation, our system requires no such participation. We envision that ISPs will willingly deploy Telex stations for a number of reasons, including idealism, goodwill, public relations, or financial incentives (e.g., tax credits) provided by governments. At worst, the consequences to ISPs for participation would be depeering, but depeering a large ISP would have a greater impact on overall network performance than blocking a single website.

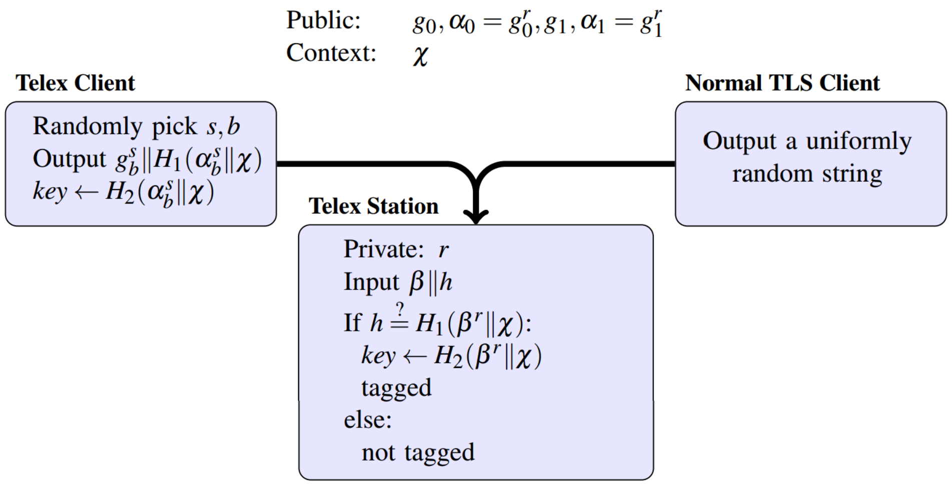
Discovery of Telex stations is a third issue. With wide enough deployment, clients could pick HTTPS servers at random. However, this behavior might divulge clients' usage of Telex, because real users don't actually visit HTTPS sites randomly. A better approach would be to opportunistically discover Telex stations by tagging flows during the course of the user's normal browsing. When a station is eventually discovered, it could provide a more comprehensive map of popular sites (where popularity is as measured with data from other Telex users) such that a Telex station is likely to be on the path between the user and the site. Even with only partial deployment, users would almost certainly discover a Telex station eventually.

4.3 Previous Work

There is a rich literature on anonymous and censorship-resistant communication for getting around this type of blocking technology, going back three decades [36]. One of the first systems explicitly proposed for combating wide-scale censorship was Infranet [52], where participating websites would discreetly provide censored content in response to steganographic requests. Infranet's designers dismissed the use of TLS because, at the time, it was not widely deployed and would be easily blocked. However, this aspect of Internet use has substantially changed since 2002 [45].

Other anticensorship approaches, including Collage [28] and Message in a Bottle [74], have leveraged using user-generated content on websites to bootstrap communication between censored users and a centrally-operated proxy. However, these designs are not intended to work with low-latency applications such as web browsing. SkypeMorph [103], FreeWave [69], CensorSpoofer [163], ScrambleSuit [169], and StegoTorus [165] are proxies or proxy-transports that attempt to mimic other protocols, such as Skype, VoIP, or HTTP in order to avoid censorship by traffic fingerprinting. However, recent work appears to suggest that such mimicry may be detectable under certain circumstances by an adversary [67, 59].

A variety of systems provide low-latency censorship resistance through VPNs or encrypted tunnels to proxies. These systems rely on servers at the edge of the network, which censors constantly try to find and block (via IP). By far, the best studied of these systems is Tor [42], which also attempts to make strong anonymity guarantees by establishing a multi-hop encrypted tunnel. Traditionally, users connect to Tor via a limited set of "entry nodes," which provide an obvious target for censors. In response, Tor has implemented bridges [147], which are a variation on Feamster et al.'s keyspace hopping [53], in which each client is told only a small subset of addresses of available proxies. While bridges provide an extra layer of protection, the arms race remains: Chinese censors now learn and block a large fraction of bridge nodes [40], possibly by using a Sybil attack [44] against the bridge address distribution system. In response, Tor has introduced obfsproxy [41], which allows users to make their entrance connection to the Tor network obfuscated with a layer of unauthenticated encryption, thwarting existing fingerprinting techniques. In order to support other obfuscating protocols, Tor builds obfsproxy on top of a generic pluggable transport architecture [118] that supports using other obfuscating or mimicry protocols, including ScrambleSuite [169], and format-transforming encryption [47], an encryption whose ciphertext can be configured to match a regular expression, such as one that generates valid HTTP messages.



Public: $g_0, \alpha_0 = g_0^r, g_1, \alpha_1 = g_1^r$
Context: χ
Telex Client
Randomly pick s, b
Output $g_b^s \| H_1(\alpha_b^s \| \chi)$
$key \leftarrow H_2(\alpha_b^s \| \chi)$
Normal TLS Client
Output a uniformly random string
Telex Station
Private: r
Input $\beta \| h$
If $h \overset{?}{=} H_1(\beta^r \| \chi)$:
$key \leftarrow H_2(\beta^r \| \chi)$
tagged
else:
not tagged

Figure 4.2: **Tag creation and detection** — Telex intercepts TLS connections that contain a steganographic tag in the ClientHello message's nonce field (normally a uniformly random string). The Telex client generates the tag using public parameters (shown above), but it can only be recognized by using the private key *r* embedded in the Telex station.

Browser-based proxies work by running a small flash proxy inside non-censored users browsers (for example, when they visit a website), and serve as short-lived proxies for censored users [55]. These rapidly changing proxies can be difficult for a censor to block in practice, though it is essentially a more fast-paced version of the traditional censor cat-and-mouse game.

4.4 Tagging

In this section, we describe how we implement the invisible tag for TLS connections, which only Telex stations can recognize. Figure 4.2 depicts the tagging scheme.

Our tags must have two properties: they must be *short*, and they must be *indistinguishable* from a uniformly random string to anyone without the private key. Someone *with* the private key should be able to examine a random-looking value and efficiently decide whether the tag is present; if so, a shared secret key is derived for use later in the protocol.

The structure of the Telex tagging system is based on Diffie-Hellman: there is a generator g of a group of prime order. Telex has a private key r and publishes a public key $\alpha = g^r$. The system uses two cryptographically secure hash functions H_1 and H_2, each salted by the current *context string* χ (see Section 4.5). To construct a tag, the client picks a random private key s, and computes g^s and $\alpha^s = g^{rs}$. If $\|$ denotes concatenation, the tag is then $g^s \| H_1(g^{rs} \| \chi)$, and the derived shared secret key is $H_2(g^{rs} \| \chi)$.

Diffie-Hellman can be implemented in many different groups, but in order to keep the tags both short and secure, we must use elliptic curve groups. Then we must ensure that, in whatever bit representation we use to transmit group elements g^s, they are indistinguishable from uniformly random strings of the same size. This turns out to be quite tricky, for three reasons:

- First, it is easy to tell whether a given (x, y) is a point on a (public) elliptic curve. Most random strings will not appear to be such a point. To work around this, we only transmit the x-coordinates of the elliptic curve points.
- Second, it is the case that these x-coordinates are taken modulo a prime p. Valid tags will never contain an x-coordinate larger than p, so we must ensure that random strings of the same length as p are extremely unlikely to represent a value larger than p. To accomplish this, we select a value of p that is only slightly less than a power of 2.
- Finally, it turns out that for any given elliptic curve, only about half of the numbers mod p are x-coordinates of points on the curve. This is undesirable, as no purported tag with an x-coordinate not corresponding to a curve point can possibly be valid. (Conversely, if a given client is observed using only x-coordinates corresponding to curve points, it is very likely using Telex.) To solve this, we use *two* elliptic curves: the original curve and a related one called the "twist". These curves have the property that every number mod p is the x-coordinate of a point on either the original curve or the twist. We will now need two generators: g_0 for the original curve, and g_1 for the twist, along with the corresponding public keys $\alpha_0 = g_0^r$ and $\alpha_1 = g_1^r$. Clients pick one pair (g_b, α_b) uniformly at random when constructing tags.

When Telex receives a candidate tag, it divides it into two parts as $\beta \| h$, according to the fixed lengths of group elements and hashes. It also determines the current context string χ. If this is a valid tag, β will be g_b^s and h will be $H_1(g_b^{rs} \| \chi)$ for some s and b. If this is not a valid tag, β and h will both be random. Thus, Telex simply checks whether $h \overset{?}{=} H_1(\beta^r \| \chi)$. This will always be true for valid tags, and will be true only with probability $2^{-\ell_{H_1}}$ for invalid tags, where ℓ_{H_1} is the bit length of the outputs of H_1. If it is true, Telex computes the shared secret key as $H_2(\beta^r \| \chi)$.

4.5 Protocol

In this section, we briefly describe the Transport Layer Security (TLS) protocol [39] and then we explain our modifications to it.

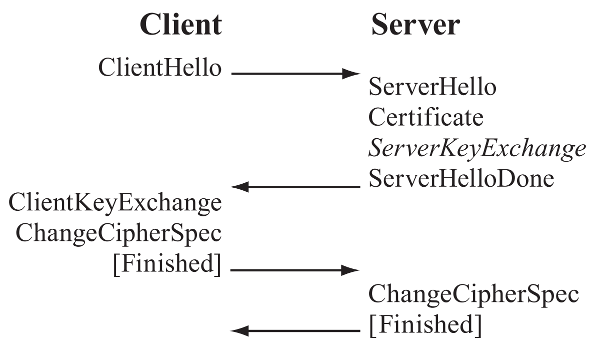


Figure 4.3: **TLS Handshake** — The client and server exchange messages to establish a shared master_secret, from which they derive cipher and MAC keys. The handshake ends with each side sending a Finished message, encrypted with the negotiated keys, that includes an integrity check on the entire handshake. The ServerKeyExchange message may be omitted, depending on the key exchange method in use.

4.5.1 Overview of TLS

TLS provides a secure channel between a client and a server, and consists of two subprotocols: the handshake protocol and the record protocol. The handshake protocol provides a mechanism for establishing a secure channel and its parameters, including shared secret generation and authentication. The record protocol provides a secure channel based on parameters established from the handshake protocol.

During the TLS handshake, the client and server agree on a cipher suite they will use to communicate, the server authenticates itself to the client using asymmetric certificates (such as RSA), and cryptographic parameters are shared between the server and client by means of a key exchange algorithm. While TLS supports several key exchange algorithms, in this chapter, we will focus on the Diffie-Hellman key exchange.

Figure 4.3 provides an outline of the TLS handshake. We describe each of these messages in detail below:

ClientHello contains a 32-byte nonce, a session identifier (0 if a session is not being resumed), and a list of supported cipher suites. The nonce consists of a 4-byte Unix timestamp, followed by a 28-byte random value.

ServerHello contains a 32-byte nonce formed identically to that in the ClientHello as well as the server's choice of one of the client's listed cipher suites.

Certificate contains the X.509 certificate chain of the server which authenticates the server to the client.

ServerKeyExchange provides the parameters for the Diffie-Hellman key exchange. These parameters include a generator g, a large prime modulus p_{DH}, a server public key, and a signature. As per the Diffie-Hellman key exchange, the server public key is generated by computing $g^{s_{priv}} \bmod p_{DH}$, where s_{priv} is a large random number generated by the server. The signature consists of the RSA signature (using the server's certificate private key) over the MD5 and SHA-1 hashes of the client and server nonces, and previous Diffie-Hellman parameters.

ServerHelloDone is an empty record, used to update the TLS state on the receiving (i.e., client) end.

ClientKeyExchange contains the client's Diffie-Hellman parameter (the client public key generated by $g^{c_{priv}} \bmod p_{DH}$).

ChangeCipherSpec alerts the server that the client's records will now be encrypted using the agreed upon shared secret. The client finishes its half of the handshake protocol with an encrypted Finished message, which verifies the cipher spec change worked by encrypting a hash of all previous handshake messages.

4.5.2 Telex handshake

The Telex handshake has two main goals: first, the censor should not be able to distinguish it from a normal TLS handshake; second, it should position the Telex station as a man-in-the-middle on the secure channel. We now describe how the Telex handshake deviates from a standard TLS handshake.

Client setup The client selects an uncensored HTTPS server located outside the censor's network (canonically, https://NotBlocked.com) and resolves its hostname to find `server_ip`. This server may be completely oblivious to the anticensorship system. The client refers to its database of Telex stations' public keys to select the appropriate key $P = (\alpha_0, \alpha_1)$ for this session. We leave the details of selecting the server and public key for future work.

ClientHello message The client generates a fresh tag τ by applying the algorithm specified in Section 4.4, using public key P and a context string composed of:

$$\texttt{server\_ip} \| \texttt{UNIX\_timestamp} \| \texttt{TLS\_session\_id}$$

This yields a 224-bit tag τ and a 128-bit shared secret key k_{sh}. Additional explanation of our tag implementation can be found in the Appendix of the original paper [175]. The client initiates a TCP connection to `server_ip` and starts the TLS handshake. As in normal TLS, the client sends a ClientHello message, but, in place of the 224-bit random value, it sends τ.

(Briefly, the tag construction ensures that the Telex station can use its private key to efficiently recognize τ as a valid tag and derive the shared secret key k_{sh}, and that, without the private key, the distribution of τ values is indistinguishable from uniform; see Section 4.4.)

If the path from the client to `server_ip` passes through a link that a Telex station is monitoring, the station observes the TCP handshake and ClientHello message. It extracts the nonce and applies the tag detection algorithm specified in Section 4.4 using the same context string and its private key. If the nonce is a genuine tag created with the correct key and context string, the Telex station learns k_{sh} and continues to monitor the handshake. Otherwise, with overwhelming probability, it rejects the tag and stops observing the connection.

Certificate validation The server responds by sending its X.509 certificate and, if necessary, key exchange values. The client verifies the certificate using the CA certificates trusted by the user's browser. It additionally checks the CA at the root of the certificate chain against a whitelist of CAs trusted by the anticensorship service. If the certificate is invalid or the root CA is not on the whitelist, the client proceeds with the handshake but aborts its Telex invocation by strictly following the TLS specification and sending an innocuous application-layer request (e.g., `GET / HTTP/1.1` for HTTPS).[2]

Key exchange At this point in the handshake, the client participates in the key exchange to compute a master secret shared with the server. We modify the key exchange in order to "leak" the negotiated key to the Telex station. Several key exchange algorithms are available. For example, in RSA key exchange, the client generates a random 46-byte master key and encrypts it using the server's public key. Alternatively, the client and server can participate in a Diffie-Hellman key exchange to derive the master secret.

The Telex client, rather than generating its key exchange values at random, seeds a secure PRG with k_{sh} and uses its output for whatever randomness is required in the key exchange algorithm (e.g., the Diffie-Hellman exponent). If a Telex station has been monitoring the connection to this point, it will know all the inputs to the client's key exchange procedure: it will have observed the server's key exchange parameter and computed the client's PRG seed k_{sh}. Using this information, the Telex station simulates the client and simultaneously derives the same master secret.

Handshake completion If a Telex station is listening, it attempts to decrypt each side's Finished message. The station should be able to use the master secret to decrypt them

[2]Both the additional root CA whitelist and the browser list need to be checked; the censor may control a CA that is commonly whitelisted by browsers, and the root CA whitelist may contain entries that are trusted by one browser but not another.

correctly and verify that the hashes match its observations of the handshake. If either hash is incorrect, the Telex station stops observing the connection. Otherwise, it switches roles from a passive observer to a man-in-the-middle. It forges a TCP RST packet from the client to NotBlocked.com, blocks subsequent messages from either side from reaching the remote end of the connection, and assumes the server's role in the unbroken TCP/TLS connection with the client.

Session resumption Once a client and server have established a session, TLS allows them to quickly resume or duplicate the connection using an abbreviated handshake. Our protocol can support this too, allowing the Telex station to continue its role as a man-in-the-middle.

The station remembers `key` and `session_id` by the server, for sessions it successfully joined. A client attempts to resume the session on a new connection by sending a ClientHello message containing the `session_id` and a fresh tag τ', which Telex can observe and verify if it is present. If the server agrees to resume the session, it responds with a ServerHello message and a Finished message encrypted with the original master secret. The client then sends its own Finished message encrypted in the same way, which confirms that it knows the original master secret. The Telex station checks that it can decrypt and verify these messages correctly, then switches into a man-in-the-middle role again.

4.6 Security Analysis

In this section, we analyze Telex's security under the threat model described in Section 4.2.1.

4.6.1 Passive attacks

First, we consider a passive censor who is able to observe arbitrary traffic within its network. For this censor to detect that a client is using Telex, it must be able to distinguish normal TLS flows from Telex flows.

Telex deviates from a normal TLS handshake in the client's nonce (sent in the ClientHello message) and in the client's key exchange parameters. In Section 4.4, we showed that an attacker cannot distinguish a Telex tag from a truly random string with more than a negligible advantage. This means that a client's tagged nonce (using Telex) is indistinguishable from a normal TLS random nonce. Likewise, the Telex-generated key exchange parameters are the output of a secure PRG; they are not distinguishable from truly random strings as a direct result of the security of the PRG.

During the TLS record protocol, symmetric cryptography is used between the Telex station and the client. A censor will be unable to determine the contents of this encrypted channel, as in normal TLS, and will thus be unable to distinguish between a Telex session and a normal TLS session from the cryptographic payload alone.

Stream cipher weakness TLS supports several stream cipher modes for encrypting data sent over the connection. Normally, the key stream is used once per session, to avoid vulnerability to a reused key attack. However, the Telex station and NotBlocked.com use the same shared secret when sending data to the client, so the same key stream is used to encrypt two different plaintexts. An attacker (possibly different from the censor) with the ability to receive both of the resulting ciphertexts can simply XOR them together to obtain the equivalent of the plaintexts XORed together. To mitigate this issue, Telex sends a TCP RST to NotBlocked.com to quickly stop it from returning data. In addition, our implementation uses a block cipher in CBC mode, for which TLS helps mitigate these issues further by providing for the communication of a random per-record IV.

We note that an adversary in position to carry out this attack (such as one surrounding the Telex station) already has the ability to detect the client's usage of Telex, as well as the contents of the connection from Telex to Blocked.com.

Traffic analysis A sophisticated adversary might attempt to detect a use of Telex by detecting anomalous patterns in connection count, packet size, and timing. Previous work shows how these characteristics can be used to fingerprint and identify specific websites being retrieved over TLS [66]. However, this kind of attack would be well beyond the level of sophistication observed in current censors [61]. We outline a possible defense against traffic analysis in Section 5.9.

4.6.2 Active attacks

Our threat model also allows the censor to attempt a variety of active attacks against Telex. The system provides strong defenses against the most practical of these attacks.

Traffic manipulation The censor might attempt to modify messages between the client and the Telex station, but Telex inherits defenses against this from TLS. For example, if the attacker modifies any of the parameters in the handshake messages, the client and Telex station will each detect this when they check the MACs in the Finished messages, which are protected by the shared secret of the TLS connection. Telex will then not intercept the connection, and the NotBlocked.com server will respond with a TLS error. Widescale manipulation of TLS handshakes or payloads would disrupt Telex; however, it would also interfere with the normal operation of TLS websites.

Tag replay The censor might attempt to use various replay attacks to detect Telex usage. The most basic of these attacks is for the censor to initiate its own Telex connection and reuse the nonce from a suspect connection; if this connection receives Telex service, the censor can conclude that the nonce was tagged and the original connection was a Telex request.

Our protocol prevents this by requiring the client to prove to the Telex station that it knows the shared secret associated with the tagged nonce. We achieve this by using the shared secret to derive the key exchange parameter, as described in Section 4.5. In particular, consider the encrypted Finished message that terminates the TLS handshake. This message must be encrypted using the freshly negotiated key (or else the TLS server will hang up), so it cannot simply be replayed. Second, the key exchange parameter in use must match the shared secret in the tagged nonce, or the Telex station will not be able to verify the MAC on the Finished message. Together, these requirements imply that the client must know the shared secret.

Handshake replay This property of proving knowledge of the shared secret is only valid if the server provides fresh key exchange parameters. An attacker may circumvent this protection by replaying traffic in *both* directions across the Telex station. This attack will cause a visible difference in the first ApplicationData message received at the client, provided that either 1) Blocked.com's response is not completely static (e.g., it sets a session cookie) or 2) the original connection being replayed was an *unsuccessful* Telex connection. In either case, the new ApplicationData message will be fresh data from Blocked.com.

A partial defense against this attack is to enforce freshness of the timestamps used in both halves of the TLS handshake and prohibit nonce reuse within the window of acceptable timestamps. However, this defense fails in the case where the original connection being replayed was an unsuccessful attempt to initiate a Telex connection, because the Telex station did not see the first use of the nonce. As a further defense, we note that NotBlocked.com will likely not accept replayed packets, and the Telex station can implement measures to detect attempts to prevent replayed packets from reaching NotBlocked.com.

Ciphertext comparison The attacker is able to detect the use of Telex if they are able to receive the unaltered traffic from NotBlocked.com, in addition to the traffic they forward to the client. Though they will not be able to decrypt either of the messages, they will be able to see that the ciphertexts differ, and from this conclude that a client is using Telex. Normally, Telex blocks the traffic between NotBlocked.com and the client after the TLS handshake to prevent this type of attack.

However, it is possible for an attacker to use DNS hijacking for this purpose. The attacker hijacks the DNS entry for NotBlocked.com to point to an attacker-controlled host. The client's path to this host passes through Telex, and the attacker simply forwards traffic from this host to NotBlocked.com. Thus, the attacker is able to observe the ciphertext traffic on both sides of the Telex station, and therefore able to determine when it modifies the traffic.

Should censors actually implement this attack, we can modify Telex stations in the following way to help detect DNS hijacking until DNSSEC is widely adopted. When it observes a tagged connection to a particular server IP, the station performs a DNS lookup based on the common name observed in the X.509 certificate. This DNS lookup returns a list of IP addresses. If the server IP for the tagged connection appears in this list, the Telex station will respond to the client and proxy the connection. Otherwise, the station will not deviate from the TLS protocol, as it is possible that the censor is hijacking DNS. This may lead to false negatives, as DNS is not globally consistent for many sites, but as long as the censor has not compromised the DNS chain that the station uses, there will be no false positives. For popular sites, we could also add a whitelisted cache of IP addresses.

Since the censor controls part of the network between the client and the Telex station, it could also try to redirect the connection by other means, such as transparently proxying the connection to a censor-controlled host. In these cases, the destination IP address observed by Telex will be different from the one specified by the client. Thus, the context strings constructed by the client and Telex will differ, and Telex will not recognize the connection as tagged. This attack offers the adversary an expensive denial of service attack, but it does not allow the attacker to detect attempted use of Telex.

Denial of service A censor may attempt to deny service from Telex in two ways. First, it may attempt to exhaust Telex's bandwidth to proxy to Blocked.com. Second, it may attempt to exhaust a Telex station's tag detection capabilities by creating a large amount of ClientHello messages for the station to check. Both methods are overt attacks that may cause unwanted political backlash on the censor or even provoke an international incident. To combat the first attack, we can implement a client puzzle [80], where Telex issues a computationally intensive puzzle the client must solve before we allow proxy service. The client puzzle should be outsourced [164] to avoid additional latency that might distinguish

Telex handshakes from normal TLS handshakes. To combat the second attack, we can implement our tag checking in hardware to increase throughput if necessary.

4.7 Implementation

To demonstrate the feasibility of Telex, we implemented a proof-of-concept client and station. While we believe these prototypes are useful models for research and experimentation, we emphasize that they may not provide the performance or security of a more polished production implementation, and should be used accordingly.

4.7.1 Client

Our prototype client program, which we refer to as `telex_client`, is designed to allow any program that uses TCP sockets to connect to the Telex service without modification. It is written in approximately 1200 lines of C (including 500 lines of shared TLS utility code) and uses libevent to manage multiple connections. The user initializes `telex_client` by specifying a local port and a remote TLS server that is not blocked by the censor (e.g. NotBlocked.com). Once `telex_client` launches, it begins by listening on the specified local TCP socket. Each time a program connects to this socket, `telex_client` initiates a TLS connection to the unblocked server specified previously. Following the Telex-TLS handshake protocol (see Section 4.5.2), `telex_client` inserts a tag, generated using the scheme described in Section 4.4, into the ClientHello nonce. We modified OpenSSL to accept supplied values for the nonce as well as the client's Diffie-Hellman exponent. This 1024-bit supplied value is generated with a secure pseudorandom generator, using input k_{sh} associated with the previous hash. These changes required us to modify fewer than 20 lines of code in OpenSSL 1.0.0.

4.7.2 Station

Our prototype Telex station uses a modular design to provide a basis for scaling the system to high-speed links and to ensure reliability. In particular, it fails safely: simple failures of the components will not impact non-Telex TLS traffic. The implementation is divided into three components, which are responsible for diversion, recognition, and proxying of network flows.

Diversion The first component consists of a router at the ISP hosting the Telex station. It is configured to allow the Telex station to passively monitor TLS packets (e.g., TCP port 443) via a tap interface. Normally, the router will also forward the packets towards their destination, but the recognition and relay components can selectively command it to not forward traffic for particular flows. This allows the other components to selectively manipulate packets and then reinject them into the network. In our implementation, the router is a Linux system that uses the iptables and ipset [75] utilities for flow blocking.

Recognition During the TLS handshake, the Telex station recognizes tagged connections by inspecting the ClientHello nonces. In our implementation, the recognition subsystem reconstructs the TCP connection using the Bro Network Intrusion Detection System [112]. Bro reconstructs the application-layer stream and provides an event-based framework for processing packets. We used the Bro scripting language for packet processing (approximately 300 lines), and we added new Bro built-in functions using C++ (approximately 450 lines).

When the Bro script recognizes a TLS ClientHello message, it checks the client nonce to see whether it is tagged. (The tag checking logic is a C implementation of the algorithm described in Section 4.4.) If the nonce is tagged, we extract the shared secret associated with the tag and create an entry for the connection in a table indexed by flow. All future event handlers test whether the flow triggering the event is contained in this table, and do nothing if it is not.

The Bro script then instructs the diversion component (via a persistent TCP connection) to block the associated flow. As this does not affect the tap, our script still receives the associated packets, and the script is responsible for actively forwarding them until the TLS Finished messages are observed. This allows the Bro script to inspect each packet before forwarding it, while ensuring that any delays in processing will not cause a packet that should be blocked to make it through the router (e.g., a TLS ApplicationData packet from NotBlocked.com to the client). To derive the TLS shared secret from the key exchange, our Bro script also stores the necessary parameters from the TLS ServerKeyExchange message in the connection table.

Once it observes the server's TLS Finished handshake message, our Bro script stops forwarding packets between the client and the server (thus atomically severing traffic flow between them) and sends the connection state, which includes the TCP-level state (sequence number, TCP options, windows, etc.), the key exchange parameters, and the shared secret k_{sh} to the proxy service component. Our proof-of-concept implementation handles only the TCP timestamp, selective acknowledgements (SACK), and window scaling options, but other options could be handled similarly. Likewise, we currently only support TLS's Diffie-Hellman key exchange, but RSA and other key exchange methods could also be supported.

Proxy service The proxy service component plays the role of the TLS server and connects the client to blocked websites. Our implementation consists of a user space process called `telex_relay` and an associated kernel module, which are responsible for decapsulating TLS connection data and passing it to a local Squid proxy[140].

The `telex_relay` process is responsible for relaying data from the client to the Squid proxy, in effect spoofing the server side of the connection. We defer forwarding of the last TLS Finished message until `telex_relay` has initialized its connection state in order to ensure that all application data is observed. We implement this delay by including the packet containing TLS Finished message in the state sent from our Bro script and leaving the task of forwarding the packet to its destination to `telex_relay`, thus avoiding further synchronization between the components.

Similarly to `telex_client`, `telex_relay` is written in about 1250 lines of C (again including shared TLS utility code) and uses libevent to manage multiple connections. It reuses our modifications to OpenSSL in order to substitute our shared secret for OpenSSL's shared secret. We implement relaying of packets between the client and the Telex service straightforwardly, by registering event handlers to read from one party and write to the other using the usual `send` and `recv` system calls on the one hand and `SSL_read` and `SSL_write` on the other.

To avoid easy detection, the relay's TCP implementation must appear similar to that of the original TLS server. Ideally, `telex_relay` would simply `bind(2)` to the address of the original server and set the `IP_TRANSPARENT` socket option, which, in conjunction with appropriate firewall and routing rules for transparent proxying [150], would cause its socket to function normally despite being bound to a non-local address. This would cause the relay's TCP implementation to be identical to that of the operating system that hosts it. However, the TCP handshake has already happened by the time our Bro script redirects the connection to `telex_relay`, so we need a method of communicating the state negotiated during the handshake to the TCP implementation. Accordingly, we modified the Linux 2.6.37 kernel to add a `fake_accept` ioctl that allows a userspace application to create a seemingly connected socket with arbitrary TCP state, including endpoint addresses, ports, sequence numbers, timestamps, and windows.

4.8 Evaluation

In this section, we evaluate the feasibility of our Telex proxy prototype based on measurements of its performance.

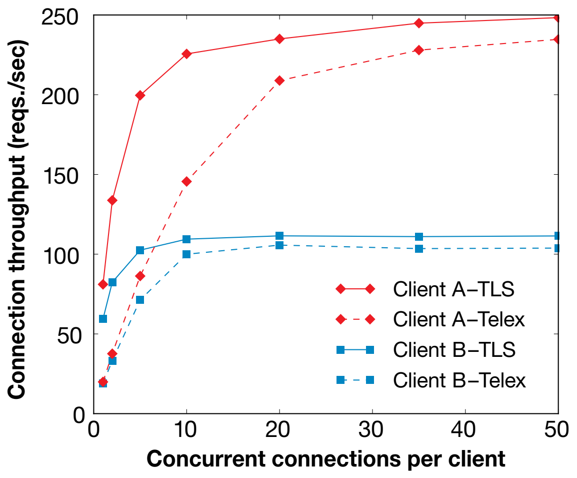


Figure 4.4: **Client Request Throughput** — We measured the rate at which two client machines could complete HTTP requests for a 1 kB page over a laboratory network, using either TLS or our Telex prototype. The prototype's performance was competitive with that of unmodified TLS.

4.8.1 Model deployment

We used a small model deployment consisting of three machines connected in a hub-and-spoke topology. Our simulated router is the hub of our deployment, and the two machines connected are the Telex station, and a web server serving pages over HTTPS and HTTP. The Telex station has a 2.93 GHz Intel Core 2 Duo E7500 processor and 2 GB of RAM. The server has a 4-core, 2.26 GHz Intel Xeon E55200 processor and 11 GB of RAM. The router has a 3.40 GHz Intel Pentium D processor and 1 GB of RAM. All of the machines in our deployment and tests are running Ubuntu Server 10.10 and are interconnected using Gigabit Ethernet.

4.8.2 Tagging performance

We evaluated our tagging implementation by generating and verifying tags in bulk using a single CPU core on the Telex station. We performed ten trials, each of which processed a batch of $100,000$ tags. The mean time to generate a batch was 18.24 seconds with a standard deviation of 0.016 seconds, and the mean time to verify a batch was 9.03 seconds with a standard deviation of 0.0083 seconds. This corresponds to a throughput of approximately 5482 tags generated per second and 11074 tags verified per second. As our TLS throughput experiments show, tag verification appears very unlikely to be a bottleneck in our system.

4.8.3 Telex-TLS performance

To compare the overhead of Telex, we used our model deployment with two additional clients connected to the router. Our primary client machine (client A) has a 2.93 GHz Intel Core 2 Duo E7500 processor and 2 GB of RAM. The secondary client machine (client B) has a 3.40 GHz Intel Pentium D processor and 2 GB of RAM. For our control, we used the Apache benchmark `ab`[10] to have each of the clients simultaneously download a static 1-kilobyte page over HTTPS. To compare to Telex, we then configured `ab` to download the same page through the `telex_client`. Because the Telex tunnel itself is encrypted with TLS, we configured `ab` to use HTTP, not HTTPS, in this latter case. For the NotBlocked.com used by `telex_client`, we used our server on port 443 (HTTPS) and for Blocked.com, we used our same server on port 80 (HTTP).

We modified `ab` to ensure that only successful connections were counted in throughput numbers and to override its use of OpenSSL's `SSL_OP_ALL` option. This option originally caused `ab` to send fewer packets than a default configuration of OpenSSL, allowing the TLS control to perform artificially better at the cost of decreased security.

We used `ab` to perform batches of 1000 connections (`ab` -n 1000); in each batch, we configured it to use a variable number of concurrent connections. We repeated each trial on our two clients (client A and client B) to get a mean connection throughput for each client.

The results are shown in Figure 4.4; the performance of the Telex tunnel lags behind that of TLS at low concurrency, but catches up at higher concurrencies. The observered performance is consistent with Telex introducing higher latency but similar throughput, which we posit is due to Telex's additional processing and network delay (e.g., execution of the `fake_accept` ioctl). Both Telex and TLS exhibit diminishing returns from more than 10 concurrent requests, and both start to plateau at 30 concurrent requests. Manual inspection of client machines' CPU utilization confirms that the tests are CPU bound by 50 concurrent connections.

4.8.4 Real-world experience

To test functionality on a real censor's network, we ran a Telex client on a PlanetLab [113] node located in Beijing and attempted connections to each of the Alexa top 100 websites [14] using our model Telex station located at the University of Michigan. As a control, we first loaded these sites without using Telex and noted apparent censorship behavior for 17 of them, including 4 from the top 10: facebook.com, youtube.com, blogspot.com and twitter.com. The blocking techniques we observed included forged RST packets, false DNS results, and destination IP black holes, which are consistent with previous findings [60]. We successfully loaded all 100 sites using Telex. We also compared the time taken to load the 83 unblocked sites with and without Telex. While this metric was difficult to measure accurately due to varying network conditions, we observed a median overhead of approximately 60%.

To approximate the user experience of a client in China, we configured a web browser on a machine in Michigan to proxy its connections over an SSH tunnel to our Telex client running in Beijing. Though each request traveled from Ann Arbor to China and back before being forwarded to its destination website (a detour of at least 32,000 km), we were able to browse the Internet uncensored, and even to watch streaming YouTube videos.

Anecdotally, three of the authors have used Telex for their daily Web browsing for about two months, from various locations in the United States, with acceptable stability and little noticeable performance degradation. The system received additional stress testing because an early version of the Telex client did not restrict incoming connections to the local host, and, as a result, one of the authors' computers was enlisted by others as an open proxy. Given the amount of malicious activity we observed before the issue was corrected, our prototype deployment appears to be robust enough to handle small-scale everyday use.

4.9 Future Work

Maturing Telex from our current proof-of-concept to a large-scale production deployment will require substantial work. In this section, we identify four areas for future improvement.

Traffic shaping An advanced censor may be able to distinguish Telex activity from normal TLS connections by analyzing traffic characteristics such as the packet and document sizes and packet timing. We conjecture that this would be difficult to do on a large scale due to the large variety of sites that can serve as NotBlocked and the disruptive impact of false positives. Nevertheless, in future work we plan to adapt techniques from prior work [66] to defend Telex against such analysis. In particular, we anticipate using a dynamic padding scheme to mimic the traffic characteristics of NotBlocked.com. Briefly, for every client request meant for Blocked.com, the Telex station would generate a real request to NotBlocked.com and use the reply from NotBlocked.com to restrict the timing and length of the reply from Blocked.com (assuming the Blocked.com reply arrived earlier). If the NotBlocked.com data arrived first, the station would send padding as a reply to the client, including a command to send a second "request" if necessary to ensure that the apparent document length, packet size, and round trip time remained consistent with that of NotBlocked.com.

Server mimicry Different service implementations and TCP stacks are easily distinguished by their observable behavior [98, Chapter 8]. This presents a substantial challenge for Telex: to avoid detection when the NotBlocked.com server and the Telex station run different software, a production implementation of Telex would need to accurately mimic the characteristics of many common server configurations. Our prototype implementation does not attempt this, and we have noted a variety of ways that it deviates from TLS servers we have tested. These deviations include properties at the IP layer (e.g. stale IP ID fields), the TCP layer (e.g. incorrect congestion windows, which is detectable by early acknowledgements), and the TLS layer (e.g. different compression methods and extensions provided by our more recent OpenSSL version). While these specific examples may themselves be trivial to fix, convincingly mimicking a diverse population of sites will likely require substantial engineering effort. One approach would be for the Telex station to maintain a set of userspace implementations of popular TCP stacks and use the appropriate one to masquerade as NotBlocked.com.

Station scalability Widescale Telex deployment will likely require Telex stations to scale to thousands of concurrent connections, which is beyond the capacity of our prototype. We plan to investigate techniques for adapting station components to run on multiple distributed machines. Clustering techniques [159] developed for increasing the scalability of the Bro IDS may be applicable.

Station placement Telex raises a number of questions related to Internet topography. How many ISPs would need to participate to provide global coverage? Short of this, where should stations be placed to optimally cover a particular censor's network? We leave accurate deployment modelling for future work.

Furthermore, we currently make the optimistic assumption that all packets for the client's connection to NotBlocked.com pass through some particular Telex station, but this might not be the case if there are asymmetric routes or other complications. Does this assumption hold widely enough for Telex to be practically deployed? If not, the system could be enhanced in future work to support cooperation among Telex stations on different paths, or to support multi-headed stations consisting of several routers in different locations diverting traffic to common recognition and relay components.

4.10 Conclusion

In this chapter, we introduced Telex, a new concept in censorship resistance. Telex demonstrates a novel censorship circumvention technique, and exploits the imperfect censoring system to allow citizens to access restricted content. By not understanding the intent of packet destinations, the censorship software can be tricked into allowing Telex traffic through. Combined with the fact that censors must make very fast decisions (that cannot be undone retroactively) on each packet, Telex can disguise proxy traffic as legitimate-looking traffic.

By moving anticensorship service from the edge of the network into the core network infrastructure, Telex has the potential to provide both greater resistance to blocking and higher performance than existing approaches. We proposed a protocol for steganographically implementing Telex on top of TLS, and we supported its feasibility with a proof-of-concept implementation. Scaling up to a production implementation will require substantial engineering effort and close partnerships with ISPs, and we acknowledge that worldwide deployment seems unlikely without government participation. However, Internet access increasingly promises to empower citizens of repressive governments like never before, and we expect censorship-resistant communication to play a growing part in foreign policy.

CHAPTER 5

Practical Censorship Circumvention

5.1 Motivation

In the last chapter, we introduced Telex, a new anticensorship technique that utilizes proxies deployed at ISPs allowing them to intercept censored users' requests to uncensored "decoy" sites, and fetch the blocked sites on the users' behalf. Telex is able to circumvent government firewalls in part because it exploits the censor's inability to inspect the intended destination of the packets that pass through it. In this case, this oversight helps adversaries that most in the western world sympathise with: citizens who wish to access blocked content. Unlike the previous chapters, where the adversaries are working toward malicious or nefarious ends (rigging elections or destroying airplanes), these citizen "adversaries" are presently limited in what they can access. By performing an act of disobedience against their unjust walls (using proxies to circumvent censorship), they are exercising their own autonomy without harming others. While the previous chapters have looked at showing how attackers *could* exploit the brittleness present in Internet voting and airport security, they have not attempted to describe exactly *how* to make such attacks reliable in practice. However, in this chapter, we will go further, and investigate ways to make proxies like Telex and others like it more *practical*.

This effort has two purposes. First, it is possible that by studying exactly how to attack law-as-code systems in practice, we can learn more about how law-as-code systems fail compared to simply hypothesizing attacks on prototype or test systems. Although we might have been technically capable of going further in the studies investigated in previous chapters, it is likely ethical considerations would prevent us from doing so. Indeed, actually rigging elections or concealing contraband past airport checkpoints might very well be illuminating from a security standpoint, but also brings potential or realised harms (such as author jail time!). However, in attacking censorship systems these ethical obligations more or less do not exist. Of course there may be other important ethical concerns, such as what content citizens can access through the proxy (such as content illegal in all countries), or whether citizens that access proxies will be punished by their government. But these can both be addressed directly by the proxy provider: service can be limited to what is legal in the proxy-hosting country, and to citizens in countries known to not punish proxy users.

The second purpose of studying this particular way to make systems like Telex more practical is because it is beneficial to the users. More than just not causing harm (as exploiting voting systems or airport security would), making a deployable version of Telex would allow people to read information and communicate freely in ways they are currently not able to. This might also ultimately encourage social progression in their countries, resulting in less repressive content policies and more transparent governments.

While these two purposes are ambitious, they do allow us to explore the more practical considerations and trade-offs that attackers face in exploiting code-as-law systems. In particular, we can learn if there are particular barriers that make attacks against such system infeasible in practice, or if the lessons we learn from previous studies remain true in practice. We also hope that by furthering this work, proxies like Telex can be deployed to the benefit of users worldwide.

Besides Telex, there are two similar proposed proxies: Cirripede [68] and Decoy Routing [81]. While the specific differences between these three proposals will be described in Section 5.2, we refer to this technique of placing proxies in ISPs generally as *end-to-middle* (E2M) *proxying*.

Deployment challenges While E2M approaches appear promising compared to traditional proxies, they face technical hurdles that have thus far prevented any of them from being deployed at an ISP. All existing schemes assume that participating ISPs will be able to selectively block connections between users and decoy sites. Unfortunately, this requires introducing new hardware in-line with backbone links, which adds latency and introduces a possible point of failure. ISPs typically have service level agreements (SLAs) with their customers and peers that govern performance and reliability, and adding in-line flow-blocking components may violate their contractual obligations. Additionally, adding such hardware increases the number of components to check when a failure does occur, even in unrelated parts of the ISP's network, potentially complicating the investigation of outages and increasing downtime. Given these risks, ISPs are reluctant to add in-line elements to their networks. In private discussions with ISPs, we found that despite being willing to assist Internet freedom in a technical and even financial capacity, none were willing to deploy existing E2M technologies due to these potential operational impacts.

Furthermore, our original E2M proposal, Telex, assumes that the ISP sees traffic in both directions, client-decoy and decoy-client. While this might be true when the ISP is immediately upstream from the decoy server, it does not generally hold farther away. IP flows are often asymmetric, such that the route taken from source to destination may be different from the reverse path. This asymmetry limits an ISP to observing only one side of a connection. The amount of asymmetry is ISP-dependent, but tier-2 ISPs typically see lower amounts of asymmetry (around 25% of packets) than tier-1s, where up to 90% of packets can be part of asymmetric flows [173]. This severely constrains where in the network E2M schemes that require symmetric flows can be deployed.

Our approach In this chapter, we propose TapDance, a novel end-to-middle proxy approach that removes these obstacles to deployment at the cost of a moderate increase in its susceptibility to active attacks by the censor. TapDance is the first E2M proxy that works without an inline-blocking or redirecting element at an ISP. Instead, our design requires only a passive tap that observes traffic transiting the ISP and the ability to inject new packets. TapDance also includes a novel connection tagging mechanism that embeds steganographic tags into the ciphertext of a TLS connection. We make use of this to allow the system to support asymmetric flows and to efficiently include large steganographic payloads in a single packet.

Although TapDance appears to be more feasible to deploy than previous E2M designs, this comes with certain tradeoffs. As we discuss in Section 5.5, there are several active attacks that a censor could perform on live flows in order to distinguish TapDance connections from normal traffic. We note that each of the previous E2M schemes is also vulnerable to at least some active attacks. As a potential countermeasure, we introduce *active defense* mechanisms, which utilize E2M's privileged vantage point in the network to induce false positives for the attacker.

Even with these tradeoffs, TapDance provides a realistic path to deployment for E2M proxy systems. Given the choice between previous schemes that appear not to be practically fieldable and our proposal, which better satisfies the constraints of real ISPs but requires a careful defense strategy, we believe TapDance is the more viable route to building anticensorship into the Internet's core.

Organization Section 5.2 reviews the three existing E2M proposals. Section 5.3 introduces our chosen ciphertext steganography mechanism, and Section 5.4 explains the rest of the TapDance construction. In Section 5.5, we analyze the security of our scheme and propose active defense strategies. In Section 5.6, we compare TapDance to previous E2M designs. We describe our proof-of-concept implementation in Section 5.7 and evaluate its performance in Section 5.8. We discuss future work in Section 5.9 and related work in Section 5.10, and we conclude in Section 5.11.

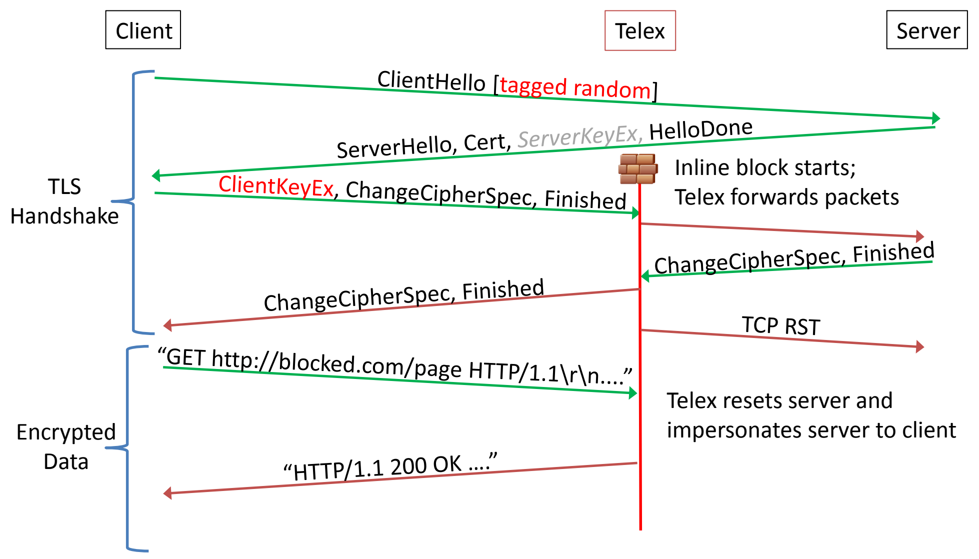

Client
Telex
Server
ClientHello [tagged random]
ServerHello, Cert, ServerKeyEx, HelloDone
Inline block starts;
Telex forwards packets
ClientKeyEx, ChangeCipherSpec, Finished
TLS
Handshake
ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
TCP RST
"GET http://blocked.com/page HTTP/1.1\r\n...."
Telex resets server and
impersonates server to client
Encrypted
Data
"HTTP/1.1 200 OK ...."

Figure 5.1: **Telex End-to-Middle Scheme** — To establish communication with an ISP-deployed Telex station, the client performs a TLS connection with an unblocked decoy server. In the TLS ClientHello message, it replaces the random nonce with a public-key steganographic tag that can be observed by the Telex station at the on-path ISP outside the censored country. When the station detects this tag with its private key, it blocks the true connection using an inline-blocking component and forwards packets for the remainder of the handshake. Once the handshake is complete, Telex stops forwarding and begins to spoof packets from the decoy server in order to communicate with the client. While here we only show the details of Telex, all of the first generation ISP proxies (Telex, Cirripede, and Decoy Routing) are similar in architecture; we note differences in Section 2.2.

The material in this chapter is adapted from "TapDance: End-to-middle anticensorship without flow blocking" by Eric Wustrow, Colleen M. Swanson, and J. Alex Halderman, which originally appeared in *Proceedings of 23rd USENIX Security Symposium*, August 2014.

5.2 Review of Existing E2M Protocols

There are three original publications on end-to-middle proxying: Telex [175], Decoy Routing [81], and Cirripede [68]. The designs for these three systems are largely similar, although some notable differences exist. Figure 5.1 show the Telex scheme, as one example.

In each design, a client wishes to reach a censored website. To do so, the client creates an encrypted connection to an unblocked *decoy* server, with the connection to this server passing through a cooperating ISP (outside the censored country) that has deployed an *ISP station*. The decoy can be any server and is oblivious to the operation of the anticensorship system. The ISP station determines that a particular client wishes to be proxied by recognizing a *tag*. In Telex, this is a public-key steganographic tag placed in the random nonce of the ClientHello message of a Transport Layer Security (TLS) connection [39]. In Cirripede, users register their IP address with a registration server by making a series of TCP connections, encoding a similar tag in the initial sequence numbers (ISNs). In Decoy Routing, the tag is placed in the TLS client nonce as in Telex, but the client and the ISP station are assumed to have a shared secret established out of band.

In both Telex and Cirripede, the tag consists of an elliptic curve Diffie-Hellman (ECDH) public key point and a hash of the ECDH secret shared with the ISP station. In Decoy Routing, the tag consists of an HMAC of the previously established shared secret key, the current hour, and a per-hour sequence number. In all cases, only the station can observe this tag, using its private key or shared secret.

Once the station has determined that a particular flow should be proxied, all three designs employ an inline blocking component at the ISP to block further communication between the client and the decoy server. Telex and Decoy Routing both block only the tagged flow using an inline-blocking component. Cirripede blocks all connections from a registered client. Cirripede's inline blocking is based on the client's IP address and has a long duration, possibly making it easier to implement than the flow-based blocking used in Telex and Decoy Routing.

After the TLS handshake has completed and the client-server communication is blocked, all three designs have the station impersonate the decoy server, receiving packets to and spoofing packets from its IP address. In Telex, the station uses the tag in the TLS client nonce to compute a shared secret with the client, which the client uses to seed its secret keys during the key exchange with the decoy server. Using this seed and the ability to observe both sides of the TLS handshake, Telex derives the master secret under which the TLS client-server communication is encrypted, and continues to use this shared secret between station and client. In Cirripede and Decoy Routing, the station changes the key stream to be encrypted under the secret exchanged during registration (Cirripede) or previously shared (Decoy Routing).

Changing the communication to a new shared secret opens Cirripede and Decoy Routing to replay and preplay attacks by the adversary. If an adversary suspects a user is accessing these proxies, it can create a new connection that replays parts from the suspected connection and receive confirmation that a particular flow uses the proxy. For example, in Decoy Routing, the adversary can simply use the suspected connection's TLS client nonce in a new connection and send a request. If the first response cannot be decrypted with the client-server shared secret, it confirms that the particular nonce was tagged. For Cirripede, a similar replay of the tagged TCP SYN packets will register the adversary's client, and a connection to the decoy server over TLS will confirm this: if the adversary can decrypt the TLS response with the established master secret, the adversary is not registered with Cirripede, indicating that the TCP SYN packets were not a secret Cirripede tag. Otherwise, if the adversary cannot decrypt the response, this indicates that the SYN packets were indeed a Cirripede tag.

Telex is not vulnerable to either of these attacks, because the client uses the client-station shared secret to seed its half of the key exchange. This allows the station to also compute the client-server shared master secret and verify that the client has knowledge of the client-server shared secret by verifying the TLS finished messages. If an adversary attempted to replay the client random in a new connection, Telex would be able to determine that the user (in this case, the adversary) did not have knowledge of the client-station shared secret, because the user did not originally generate the Diffie-Hellman tag. Thus, Telex is unable to decrypt and verify the TLS finished messages as expected, and will not spoof messages from the server.

Both Cirripede and Decoy Routing function in the presence of asymmetric flows. In Cirripede, the station only needs to observe communication from the client to the decoy server in order to establish its shared secret with the client. In Decoy Routing, the client sends any missing information (i.e., information contained in messages from the server to the client) via another covert channel. In contrast, Telex's approach does not handle asymmetric paths, as the station needs to see both sides of the communication in order to learn the client-server shared master secret.

Unlike any of the existing schemes, TapDance functions without an inline blocking component, potentially making it much easier to deploy at ISPs. Unlike Telex, it supports asymmetric flows, but in doing so it sacrifices some of Telex's resistance to active attacks. We defer a complete comparison between TapDance and the first-generation E2M schemes until Section 5.6, after we have introduced the details of the system.

5.3 Ciphertext Covert Channel

Previous E2M covert channels have been limited in size, forcing implementations to use small payloads or several flows in order to steganographically communicate enough information to the ISP station. However, because TapDance does not depend on inline flow-blocking and must work with asymmetric flows, we need a way to communicate the client's request directly to the TapDance station while maintaining a valid TLS session between the client and the decoy server. We therefore introduce a novel technique, *chosen-ciphertext steganog-*

raphy, which allows us to encode a much higher bandwidth steganographic payload in the ciphertexts of legitimate (i.e., censor-allowed) TLS traffic.

The classic problem in steganography is known as the *prisoners' problem*, formulated by Simmons [134]: two prisoners, Alice and Bob, wish to send hidden messages in the presence of a jailer. These messages are disguised in legitimate, public communication between Alice and Bob in such a way that the jailer cannot detect their presence. Many traditional steganographic techniques focus on embedding hidden messages in non-uniform cover channels such as images or text [16]; in the network setting, each layer of the OSI model may provide potential cover traffic [65] of varying bandwidths. To avoid detection, these channels must not alter the expected distribution of cover traffic [107]. In addition, use of header fields in network protocols for steganographic cover limits the carrying capacity of the covert channel.

We observe it is possible for the sender to use stream ciphers and CBC-mode ciphers as steganographic channels. This allows a sender Alice to embed an arbitrary hidden message to a *third party*, Bob, inside a *valid* ciphertext for Cathy. That is, Bob will be able to extract the hidden message and Cathy will be able to decrypt the ciphertext, without alerting outside entities (or, indeed, Cathy, subject to certain assumptions) to the presence of the steganographic messages.

Moreover, through this technique, we can place limited constraints on the plaintext (such as requiring it be valid base64 or numeric characters), while encoding arbitrary data in the corresponding ciphertext. This allows us to ensure not only that Cathy can decrypt the received ciphertext, but also that the plaintext is consistent with the protocol used. Note that this is a departure from the original prisoners' problem, as we assume Alice is allowed to securely communicate with Cathy, so long as this communication looks legitimate to outside entities.

As our technique works both with stream ciphers and CBC-mode ciphers, which are the two most common modes used in TLS [91], we will use this building block to encode steganographic tags and payloads in the ciphertext of TLS requests.

5.3.1 Chosen-Ciphertext Steganography

To describe our technique, we start with a stream cipher in counter mode. The key observation is that counter mode ciphers, even with authentication tags, have ciphertexts that are *malleable* from the perspective of the sender, Alice. That is, stream ciphers have the general property of *ciphertext malleability*, in that flipping a single bit in the ciphertext flips a single corresponding bit in the decrypted plaintext. Alice can likewise change bits in

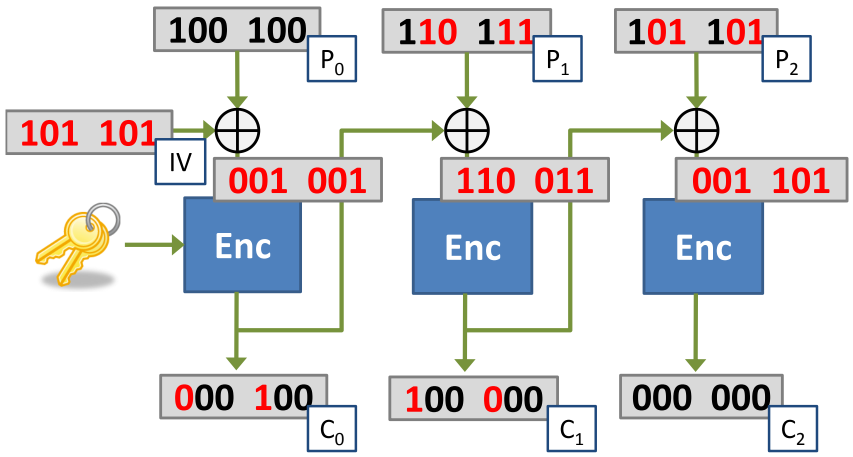


Figure 5.2: **CBC Chosen Ciphertext Example**—In this example, bits chosen by the encoding are in black, while bits "forced" by computation are red. For example, we choose all 6-bits to be 0 in the last ciphertext block. This forces the block's intermediary to be "forced" to a value beyond our control; in this case 001101. To obtain this value, we can choose a mixture of bits in the plaintext, which forces the corresponding bits in the previous ciphertext block. In this example, we choose the plaintext block to be of the form $1xx1xx$, allowing us to choose 4-bits in the ciphertext, which we choose to be 0s. Thus, the ciphertext has the form $x00x00$. We solve for the unknown bits in the ciphertext and plaintext ($1xx1xx \oplus x00x00 = 001101$) to fill in the missing "fixed" values. We can repeat this process backward until the first block, where we simply compute the IV in order to allow choosing all the bits in the first plaintext block.

the plaintext to effect specific bits in the corresponding ciphertext. Since Alice knows the keystream for the stream cipher, she can choose an arbitrary string that she would like to appear in the ciphertext, and compute (decrypt) the corresponding plaintext. Note that this does not invalidate the MAC or authentication tag used in addition to this cipher, because Alice first computes a valid plaintext, and then encrypts and MACs it using the standard library, resulting in ciphertext that contains her chosen steganographic data.

Furthermore, Alice can "fix" particular bits in the plaintext and allow the remaining bits to be determined by the data encoded in the ciphertext. For example, Alice could require that each plaintext byte starts with 5 bits set to 00110, and allow the remaining 3 bits to be chosen by the ciphertext. In this way, the plaintext will always be an ASCII character from the set "01234567" and the ciphertext has a steganographic "carrying capacity" to encode 3 bits per byte.

While it seems intuitive that Alice can limit plaintext bits for stream ciphers, it may not be as intuitive to see how this is also possible for CBC-mode ciphers. However, while the ciphertext malleability of stream ciphers allows Alice partial control over the resulting plaintext, we show that it is also possible to use this technique in other cipher modes, with equal control over the plaintext values.

In CBC mode, it is possible to choose the value of an arbitrary ciphertext block (e.g., C_2), and decrypt it to compute an intermediary result. This intermediary result must also be the result of the current plaintext block (P_2) xored with the previous ciphertext block (C_1) in order to encrypt to the chosen ciphertext value. This means that, given a ciphertext block, we can choose either the plaintext value (P_2), or the previous ciphertext block (C_1), and compute the other. However, we can also choose a mixture of the two; that is, for each bit we pick in the plaintext, we are "forced" to choose that corresponding bit in the previous plaintext block and vise-versa. Choosing any bits in a ciphertext block (C_1) will force us to repeat this operation for the previous plaintext block (P_1) and twice previous ciphertext block (C_0). We can choose to pick the value of plaintext blocks (fixing the corresponding ciphertext blocks), all the way back to the first plaintext block, where we are left to decide if we want to choose the value of the first plaintext block or the Initialization Vector (IV) value. At this point, fixing the IV is the natural choice, as this leaves us greater control over the first plaintext block. Figure 5.2 shows an example of this backpropagation, encoding a total of 4-bits per 6-bit ciphertext block (plus a full final block).

This scheme allows us to restrict plaintexts encrypted with CBC to the same ASCII range as before, while still allowing us to encode arbitrary-length messages in the ciphertext.

While the sender can encode any value in the ciphertext in this manner, we do not wish to change the expected ciphertext distribution. The counter and CBC modes of encryption both satisfy indistinguishability from random bits [125], so encoding anything that is distinguishable from a uniform random string would allow third parties (e.g., a network censor) to detect this covert channel. To prevent this, Alice encrypts her hidden message if necessary, using an encryption scheme that produces ciphertexts indistinguishable from random bits. The resulting ciphertext for Bob is then encoded in the CBC or stream-cipher ciphertext as outlined above. To an outside adversary, this resulting "ciphertext-in-ciphertext" should still be a string indistinguishable from random, as expected.

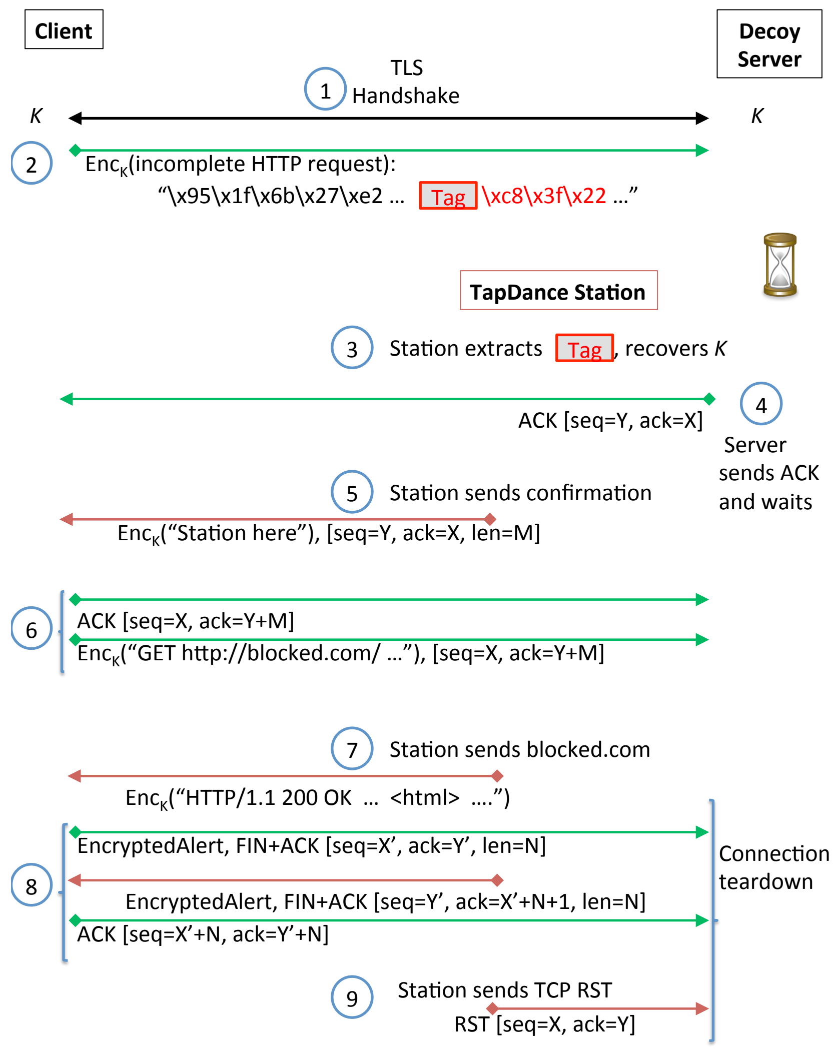

Client
Decoy Server
1 TLS Handshake
K
K
2 Enc_K(incomplete HTTP request):
"\x95\x1f\x6b\x27\xe2 ... Tag \xc8\x3f\x22 ..."
TapDance Station
3 Station extracts Tag, recovers K
ACK [seq=Y, ack=X]
4 Server sends ACK and waits
5 Station sends confirmation
Enc_K("Station here"), [seq=Y, ack=X, len=M]
6 ACK [seq=X, ack=Y+M]
Enc_K("GET http://blocked.com/ ..."), [seq=X, ack=Y+M]
7 Station sends blocked.com
Enc_K("HTTP/1.1 200 OK ... <html> ....")
8 EncryptedAlert, FIN+ACK [seq=X', ack=Y', len=N]
EncryptedAlert, FIN+ACK [seq=Y', ack=X'+N+1, len=N]
ACK [seq=X'+N, ack=Y'+N]
Connection teardown
9 Station sends TCP RST
RST [seq=X, ack=Y]

Figure 5.3: **TapDance Overview** — (1) The client performs a normal TLS handshake with an unblocked decoy server, establishing a session key K. (2) The client sends an *incomplete* HTTP request through the connection and encodes a steganographic tag in the *ciphertext* of the request, using a novel encoding scheme (Section 5.4.2). (3) The TapDance station observes and extracts the client's tag, and recovers the client-server session secret K. (4) The server sends a TCP ACK message in response to the incomplete HTTP request and waits for the request to be completed or until it times out. (5) The station, meanwhile, spoofs a response to the client from the decoy server. This message is encrypted under K and indicates the station's presence to the client. (6) The client sends a TCP ACK (for the spoofed data) and its real request (blocked.com). The server ignores both of these, because the TCP acknowledgment field is higher than the server's TCP SND.NXT. (7) The TapDance station sends back the requested page (blocked.com) as a spoofed response from the server. (8) When finished, the client and TapDance station simulate a standard TCP/TLS authenticated shutdown, which is again ignored by the true server. (9) After the connection is terminated by the client, the TapDance station sends a TCP RST packet that is valid for the server's SND.NXT, silently closing its end of the connection before its timeout expires.

5.4 TapDance Architecture

5.4.1 Protocol Overview

The TapDance protocol requires only a passive network tap and traffic injection capability, and is carefully designed to work even if the station is unable to observe communication between the decoy server and the client. To accomplish this, we utilize several tricks gleaned from a close reading of the TCP specification [116] to allow the TapDance station to impersonate the decoy server without blocking traffic between client and server.

Figure 5.3 gives an overview of the TapDance protocol. In the first step, the client establishes a normal TLS connection to the decoy web server. Once this handshake is complete, the client and decoy server share a master secret, which they use to generate encryption keys, MAC keys, and initialization vector or sequence state.

The TapDance protocol requires the client to leak knowledge of the client-server master secret, thereby allowing the station to use this shared secret to encrypt all communications. The client encodes the master secret as part of a steganographic tag visible only to the TapDance station. This tag is hidden in an *incomplete* HTTP request sent to the decoy server through the encrypted channel. Since this request is incomplete, the decoy server will not respond with data to the client; this can be accomplished, for example, by simply withholding the two consecutive line breaks that mark the end of an HTTP request. The decoy server will acknowledge this data only at the TCP level by sending a TCP ACK packet and will then wait for the rest of the client's incomplete HTTP request until it times out. As shown in Figure 5.5, our evaluation reveals that most TLS hosts on the Internet will leave such incomplete request connections open for at least 60 seconds before sending additional data or closing the connection.

When the TapDance station observes this encrypted HTTP request, it is able to extract the tag (and hence the master secret), as discussed in detail in Section 5.4.2. The station then spoofs an encrypted response from the decoy server to the client. This message acts as confirmation for the client that the TapDance station is present. In particular, this message is consistent with a pipelined HTTPS connection, so by itself does not indicate that TapDance is in use.

At the TCP level, the client acknowledges this spoofed data with a TCP ACK packet, and because there is no inline-blocking between it and the server, the ACK will reach the server. However, because the acknowledgment number is above the server's $SND.NXT$, the server will not respond. Similarly, if the client responds with additional data, the acknowledgment field contained in those TCP packets will also be beyond what the server has sent. This allows the TapDance station to continue to impersonate the server, acknowledging data the client sends, and sending its own data in response, without interference from the server itself.

5.4.2 Tag Format

In TapDance, we rely on elliptic curve Diffie-Hellman to agree on a per-connection shared secret between the client and station, which is used to encrypt the steganographic tag payload. The tag consists of the client's connection-specific elliptic curve public key point ($Q = eG$), encoded as a string indistinguishable from uniform, followed by a variable-length encrypted payload used to communicate the client-server TLS master secret (and intent for proxying) to the station.

In order to properly disguise the client's elliptic curve point, we use Elligator 2 [23] over Curve25519 [22]. Elligator 2 is an efficient encoding function that transforms, for certain types of elliptic curves, exactly half of the points on the curve to strings that are indistinguishable from uniform random strings.

The client uses the TapDance station's public key point ($P = dG$) and its own private key (e) to compute an ECDH shared secret with the station ($S = eP = dQ$), which is used to derive the payload encryption key. The encrypted payload contains an 8-byte magic value used by the station to detect successful decryption, the client and server random nonces, and the client-server master secret of the TLS connection. With this payload, typically contained in a single packet from the client, the station is able to derive the TLS master secret between client and server.

We insert the tag, composed of the encoded point and encrypted payload, into the ciphertext of the client's incomplete request to the server using the chosen ciphertext steganographic channel described in Section 5.3. In order to avoid the server generating unwanted error messages, we maintain some control over the plaintext that the server receives using the plaintext-limiting technique as described in Section 5.3. Specifically, we split the tag into 6-bit chunks and encode each chunk in the low order bits of a ciphertext byte. This allows the two most significant bits to be chosen freely in the plaintext (i.e. not decided by the decryption of the tag-containing ciphertext). We choose these two bits so that the plaintext always falls within the ASCII range 0x40 to 0x7f. We verified that Apache was capable of handling this range of characters in a header line without triggering an error.

5.5 Security Analysis

Our threat model is similar to that of previous end-to-middle designs. We assume an adversarial censor that can observe, alter, block, or inject network traffic within their domain or geographic region (i.e., country) and may gain access to foreign resources, such as VPNs or private servers, by leasing them from providers. Despite control over its network infrastructure, however, we assume the censor does not have control over end-users' computers, such as the ability to install arbitrary programs or Trojans.

The censor can block its citizens' access to websites it finds objectionable, proxies, or other communication endpoints it chooses, using IP blocking, DNS blacklists, and deep-packet inspection. We assume the censor uses blacklisting to block resources and that the censor does not wish to block legitimate websites or otherwise cut themselves off from the rest of the Internet, which may inhibit desirable commerce or communication. In addition, we assume that the censor allows end-to-end encrypted communication, specifically TLS communication. As websites increasingly support HTTPS, censors face increasing pressures against preventing TLS connections [45].

While the threat model for TapDance is similar to those assumed by prior end-to-middle schemes, our fundamentally new design has a different attack surface than the others. We perform a security analysis of TapDance and compare it to the previous generation designs, focusing on the adversarial goal of distinguishing normal TLS connections from TapDance connections. In particular, we do not attempt to hide the deployment locations of the TapDance stations themselves.

5.5.1 Passive Attacks

TLS handshake TLS allows implementations to support many different extensions and cipher suites. As a result, implementations can be easy to differentiate based on the ciphers and extensions they claim to support in their ClientHello or ServerHello messages. In order to prevent this from being used to locate suspicious implementations, our proxy must blend in to or mimic another popular client TLS implementation. For example, we could support the same set of ciphers and extensions as Chrome for the user's platform. Currently, our client mimics Chrome's cipher suite list for Linux.

Cryptographic attacks A computationally powerful adversary could attempt to derive the station's private key from the public key. However, our use of ECC Curve25519 should resist even the most powerful computation attacks using known discrete logarithm algorithms. The largest publicly known ECC key to be broken is only 112 bits, broken over 6 months in 2009 on a 200-PlayStation3 cluster [25]. In contrast to Telex, TapDance also supports increasing the key size as needed, as we are not limited to a fixed field size for our tag.

Forward secrecy An adversary who compromises an ISP station or otherwise obtains a station's private key can use it to trivially detect both future and previously recorded flows in order to tell if they were proxy flows. Additionally, they can use the key to decrypt the user's request (and proxy's response), learning the censored websites users have visited. To address the first problem, we can use a technique suggested in Telex [175]. The ISP station generates many private keys ahead of time and stores them in either a hardware security module or offline storage, and provides all of the public keys to the clients. Clients can then cycle through the public keys they use based on a course-grained time (e.g., hours or days). The proxy could also cycle through keys, destroying expired keys and limiting access to future ones.

To address the second problem, TapDance is compatible with existing forward-secure protocols. For example, for each new connection it receives, the TapDance station can generate a new ECDH point randomly, and establish a new shared secret between this new point and the original point sent by the client in the connection tag. The station sends its new ECDH public point to the client in its Hello message, and the remainder of the connection is encrypted under the new shared secret. This scheme has the advantage that it adds no new round trips to the scheme and only 32-bytes to the original ISP station's response.

Packet timing and length The censor could passively measure the normal round-trip time between potential servers and observe the set of packet lengths of encrypted data that a website typically returns. During a proxy connection, the round-trip time or the packet lengths of the apparent server may change for an observant censor, as the station may be closer or have more processing delay than the true server. This attack is possible on all three of the first generation E2M schemes, as detailed in [130]. However, such an attack at the application level may be difficult to carry out in practice, as larger, legitimate websites may have many member-only pages that contain different payload lengths and different processing overhead. The censor must be able to distinguish between "blind pages" it cannot confirm are part of the legitimate site and decoy proxy connections. We note that this is difficult at the application level, but TCP round-trip times may have a more consistent and distinguishable difference.

Lack of server response If the TapDance station fails to detect a client's flow, it will not respond to the client. This may appear suspicious to a censor, as the client sends a request, but there is no response at the application level from the server. This scenario could occur for three reasons. First, the censor may disrupt the path between client and TapDance station in order to cause such a timeout, using one of the active attacks below (such as the routing-around attack), in order to confirm a particular flow is attempting to use TapDance. Second, such false pickups may happen intermittently (due to ISP station malfunction). Finally, a client may attempt to find new TapDance stations by probing many potential decoy servers with tagged TLS connections. Paths that do not contain ISP stations will have suspiciously long server response times.

To address the last issue, probing clients could send complete requests and tag their requests with a secret nonce. The station could record these secret nonces, and, at a later time (out of band, or through a different TapDance station), the client can query the station for the secret nonces it sent. In this way, the client learns new servers for which the ISP station is willing to proxy without revealing the probing pattern. To address the first two problems, we could have clients commonly use servers that support long-polling HTTP push notification. In these services, normal requests can go unanswered at the application layer as long as the server does not have data to send to the client, such as in online-gaming or XMPP servers. Another defense is to have the client send complete requests that force the server to keep the connection alive for additional requests, and to have the TapDance station inject additional data *after* the server's initial response. This requires careful knowledge of the timing and length of the server's initial response, which could either be provided by active probing from the station or information given by the client.

TCP/IP protocol fingerprinting The adversary could attempt to observe packets coming from potential decoy servers and build profiles for each server, including the set of TCP options supported by the server, IP TTL values, TCP window sizes, and TCP timestamp slope and offset. If these values ever change, particularly in the middle of a connection (and only for that connection), it could be a strong indication of a particular flow using a proxy at an on-path ISP. To prevent this attack, the station also needs to build these profiles for servers, either by actively collecting this profile from potential servers, or passively observing the server's responses to non-proxy connections and extracting the parameters. Alternatively, the client can signal to the station some of the parameters. First generation schemes varied in defense for this type of attack; for example, Telex's implementation is able to infer and mimic all of these parameters from observing the servers' responses, although Telex requires a symmetric path in order to accomplish this. In theory, parameters that the adversary can measure for fingerprinting can also be measured by the station and mimicked. However, given that the adversary has only to find one distinguisher in order to succeed, server mimicry remains difficult to achieve in practice.

5.5.2 Active Attacks

TLS attacks The censor may issue fake TLS certificates from a certificate authority under its control and then target TLS sessions with a man-in-the-middle attack. While TapDance and previous designs are vulnerable to this attack, there may be external political pressure that discourages a censor from this attack, as it may be disruptive to foreign e-commerce in particular. We also argue that as the number of sites using TLS continues to increase, this attack becomes more expensive for the censor to perform without impacting performance. Finally, decoy servers that use certificate pinning or other CA-protection mechanisms such as Perspectives [166], CAge [82], or CA country pinning [138], can potentially avoid such attacks.

Packet injection Because TapDance does not block packets from the client to the true server, it is possible for the censor to inject spoofed probes from the client that will reach the server. If the censor can craft a probe that will result in the server generating a response that reveals the server's true TCP state, the censor will be able to use this response to differentiate real connections from proxy connections. While the previous designs also faced this threat [130], the censor had to inject the spoofed packet in a way that bypassed the station's ISP inline blocking element. In TapDance, there is no blocking element, and so the censor is able to simply send it without any routing tricks. An example of this attack is the censor sending a TCP ACK packet with a stale sequence number, or one for data outside the server's receive window. The server will respond to this packet with an ACK containing the server's TCP state (sequence and acknowledgment), which will be smaller than the last sequence and/or acknowledgments sent by the station.

There are a few ways to deal with this attack if the censor employs it. First, we can simply limit each proxy connection to a single request from the client and a response from the station, followed immediately by a connection close. This will dramatically increase the overhead of the system but will remove the potential for the adversary to use injected packets and their responses to differentiate between normal and proxy connections. This is because the TCP state between the station and real server will not diverge until the station has sent its response, leaving only a very small window where the censor can probe the real server for its state and get a different response.

Active defense Alternatively, in order to frustrate the censor from performing packet injection attacks, we can perform *active defense*, where the station observes active probes such as the TCP ACK and responds to them in a way that would "reveal" a proxy connection, even for flows that are not proxy connections. To the censor, this would make even legitimate non-proxy connections to the server appear as if they were proxy connections.

As an example, consider a censor that injects a stale ACK for suspected proxy connections. Connections that are actually proxy connections will respond with a stale ACK from the server, revealing the connection to the censor. However, the station could detect the original probe, and if it is not a proxied connection, respond with a stale ACK so as to make it appear to the censor as if it were. In this way, for every probe the censor makes, they will detect, sometimes incorrectly, that the connection was a proxy connection.

Replay attacks The censor could capture suspected tags and attempt to replay them in a new connection, to determine if the station responds to the tag. To specifically attack TapDance, the adversary could replay the client's tag-containing request packet after the connection has closed and observe if the station appears to send back a response. We note that both Cirripede and Decoy Routing are also vulnerable to tag replay attacks, although Telex provides some limited protection from them. To protect against duplicated tags, the station could record previous tags and refuse to respond to a repeated tag. To avoid saving all tags, the station could require clients to include a recent timestamp in the encrypted payload[1].

However, such a defense may enable a denial of service attack: the censor could delay the true request of a suspected client and send it in a different connection first. In this *preplay* version of the attack, the censor is also able to observe whether the station responds with the ClientHello message. If it does, the censor will know the suspected request contained a tag.

Denial of service The censor could attempt to exhaust the station's resources by creating many proxy connections, or by sending a large volume of traffic that the ISP station will have to check for tags using an expensive ECC function. We estimate that a single ISP station deployment of our implementation on a 16-core machine could be overwhelmed if an attacker sends approximately 1.2 Gbps of pure TLS application data packets past it. This type of attack is feasible for an attack with a small botnet, or even a few well-connected servers. Because ISPs commonly perform load balancing by flow-based hashing, we can scale our deployment linearly to multiple branches of machines and use standard intrusion detection techniques to ignore packets that do not belong to valid connections or that come from spoofed or blacklisted sources [112].

[1]The client random which is sent in the encrypted payload already contains a timestamp for the first 4 bytes

Routing around the proxy A recent paper by Schuchard et al. details a novel attack against our and previous designs [130]. In this attack, the censor is able to change local routing policy in a way that redirects outbound flows around potential station-deploying ISPs while still allowing them to reach their destinations. This prevents the ISP station from being able to observe the tagged flows and thus from being a proxy for the clients. However, Houmansadr et al. investigate the cost to the censor of performing such an attack and find it to be prohibitively expensive [70]. Although both of these papers ultimately contribute to deciding which ISPs should deploy proxies in order to be most resilient, we consider such a discussion outside our current scope.

Tunneling around the proxy A more conceptually simple attack is for the censor to transparently tunnel specific suspected flows around the ISP station. For example, the censor could rent a VPN or VPS outside the country and send specific flows through them to avoid their paths crossing the ISP station. This attack is expensive for the adversary to perform, and so could not reasonably be performed for an entire country. However, it could be performed for particular targets and combined with previous passive detection attacks to aid the censor in confirming whether particular users are tagging their flows.

Complicit servers A censor may be able to compromise, coerce, or host websites that can act as servers for decoy connections. The vantage point from a server allows them to observe incomplete requests from clients, including the plaintext that the client mangled in order to produce the tag in the ciphertext. This allows the censor to both observe specific clients using the ISP station and also disrupt use of the proxy with the particular server. There is little TapDance or previous designs can do to avoid cooperation between servers and the censor, as the two can simply compare traffic received and detect proxy flows as ones that have different data at the two vantage points. However, using this vantage point to disrupt proxy use could be detected by clients and the server avoided (and potentially notified in the case of a compromise).

5.6 Comparison

On the protocol level, TapDance bears more similarity to Telex than Cirripede, in that clients participate in TapDance on a per-connection basis, rather than participating in a separate registration phase as in Cirripede, and in that client-station communication, after the initial Diffie-Hellman handshake, is secured using the client-server master secret. In order to conserve bandwidth, our design, like both Telex and Cirripede, leverages elliptic curve

	Telex [175]	Cirripede [68]	Decoy Routing [81]	TapDance
Steganographic channel	TLS nonce	TCP ISNs	TLS nonce	TLS ciphertext
Works passively	○	○	○	●
Handles asymmetric flows	○	●	●	●
Proxies per flow	●	○	●	●
Replay attack resistant	●	○	○	○
Traffic analysis defense	○	○	○	○

Table 5.1: **Comparing E2M Schemes** — Unlike previous work, TapDance operates without an inline flow-blocking component at cooperating ISPs. However, it is vulnerable to active attacks that some previous designs resist. No E2M system yet defends against traffic analysis or website fingerprinting, making this an important area for further study.

cryptography to signal intent to use the system and to establish a shared secret between client and station.

However, TapDance exhibits several important differences from previous protocols, which has implications for both security and functionality. As discussed in Section 5.1, one of the largest challenges to deploying E2M proxies at ISPs is the inline flow-blocking component. TapDance has the singular advantage in that it allows client-server communication to continue unimpeded. In fact, our design requires only that the TapDance station be able to passively observe communication from client to server and be able to inject messages into the network; the station can be oblivious to communication passing from server to client.

The advantages of the TapDance protocol stem from its careful use of chosen-ciphertext steganography (described in Section 5.3) to hide the client's tag and the fact that a high percentage of servers ignore stale TCP-level messages. In contrast, previous proposals rely on inline blocking to prevent server-client communication, and TCP sequence numbers and TLS ClientHello random nonces to disguise the client's steganographic tag. In general, these fields are useful in steganography because these strings should be uniformly random for legitimate connections, providing a good cover for the tag that replaces them, so long as this tag is indistinguishable from random.

However, both of these fields are fixed size; each TLS nonce can be replaced with a 224-bit uniform random tag, and each TCP sequence number with only 24 bits of a tag. Cirripede, which encodes the client's tag into TCP sequence numbers, uses multiple TCP connections to convey the full tag to the station. Telex and Decoy Routing both use a single TLS nonce to encode the client's tag. Given the limited bandwidth of these covert channels, they are useful to convey only short secrets, while the rest of the payload (such as the request for a blocked website) must take place in a future packet.

TapDance, on the other hand, leverages chosen-ciphertext steganography in order to encode steganographic tags in the ciphertext of a TLS connection, without invalidating the

TLS session itself. Encoding the tag in the ciphertext has several advantages. First, the tag is no longer constrained to a fixed field size of either 24 or 224 bits, allowing us to encode more information in each tag, and use larger and more secure elliptic curves. Second, because the ciphertext is sent after the TLS handshake has completed, it is possible to encode the connection's master secret in this tag, allowing the station to decrypt the TLS session from a single packet, and without requiring the station to observe packets from the server.

In addition, TapDance takes advantage of recent work by Bernstein et al. [23], in order to disguise elliptic curve points as strings indistinguishable from uniform, namely Elligator 2. Traditional encoding of elliptic curve points is distinguishable from random for several reasons, which are outlined in detail in [23]. Telex and Cirripede address this concern by employing two closely related elliptic curves, which is less efficient than TapDance's use of Elligator 2, as the latter method requires only a single elliptic curve to achieve the same functionality.

From a security perspective, the only attacks unique to TapDance are the lack of server response and packet injection attacks. Besides these, we find our design has no additional vulnerabilities from which all previous designs were immune. While these two attacks do pose a threat to TapDance, the benefits of a practical ISP station deployment—at least as a bridge to stronger future systems—may outweigh the potential risks.

In summary, our approach obviates the need for an inline blocking element at the ISP, which is a requirement of Telex, Cirripede, and Decoy Routing, while preserving system functionality in the presence of asymmetric flows, which is an advantage over Telex. In addition, the covert channel used in TapDance is higher bandwidth than that of previous proposals and holds potential for future improvements (e.g., in terms of number of communication rounds required and flexible security levels) of client-station protocols.

5.7 Implementation

We have implemented TapDance in two parts: a client that acts as a local HTTP proxy for a user's browser, and a station that observes a packet tap at an ISP and injects traffic when it detects tagged connections. Our station code is written in approximately 1,300 lines of C, using libevent, OpenSSL, PF_RING [110], and forge_socket[2].

[2] https://github.com/ewust/forge_socket/

5.7.1 Client Implementation

Our client is written in approximately 1,000 lines of C using libevent [100] and OpenSSL [117]. The client currently takes the domain name of the decoy server as a command line argument, and for each new local connection from the browser, creates a TLS connection to the decoy server. Once the handshake completes, the client sends the incomplete response to prevent the server from sending additional data, and to encode the secret tag in the ciphertext as specified in Section 5.4.2. The request is simply an HTTP request with a valid HTTP request line, “Host” header, and an “X-Ignore” header that precedes the “garbage” plaintext that will be computed to result in the chosen tag appearing in the ciphertext. We have implemented our ciphertext encoding for AES_128_GCM [129], although it also works without modification for AES_256_GCM cipher suites. We have implemented Elligator 2 to work with Curve25519, in order to encode the client’s public point in the ciphertext as a string that is indistinguishable from uniform random. After this 32-byte encoded point, the client places a 144-byte encrypted payload. This payload is encrypted using a SHA-256 hash of the 32-byte shared secret (derived from the client’s secret and station’s public point) using AES-128 in CBC mode. We use the first 16-bytes of the shared secret hash as the key, and the last 16 bytes as the initialization vector (IV). The payload contains an 8-byte magic value, the 48-byte TLS master secret, 32-byte client random, 32-byte server random, and a 16-byte randomized connection ID that allows a client to reconnect to a previous proxy connection in case the underlying decoy connection is prematurely closed.

5.7.2 Station Implementation

Our TapDance station consists of a 16-core Supermicro server connected over a gigabit Ethernet to a mirror port on an HP 6600-24G-4XG switch in front of a well-used Tor exit node generating about 160 Mbps of traffic. The station uses PF_RING, a fast packet capture Linux kernel module, to read packets from the mirror interface. In addition to decreasing packet capture overhead, PF_RING supports flow clustering, allowing our implementation to spread TCP flow capture across multiple processes. Using this library, our station can have several processes on separate cores share the aggregate load.

For each unique flow (4-tuple), we keep a small amount of state whether we have seen an Application Data packet for the flow yet. If we have not, we verify the current packet’s TCP checksum, and inspect the packet to determine if it is an Application Data packet. If it is, we mark this flow as having been inspected, and pass the packet ciphertext to the tag extractor function. This function extracts the potential tag from the ciphertext, decoding the client’s public point using Elligator 2, generating the shared secret using Curve25519, and hashing it to get the AES decryption key for the payload. The extractor decrypts the 144-byte payload included by the client, and verifies that the first 8 bytes are the expected magic value. If it is, the station knows this is a tagged flow, and uses the master secret and nonces extracted from the encrypted payload to compute the key block, which contains encryption and decryption keys, sequence numbers or IVs, and MAC keys (if not using authenticated encryption) for the TLS session between the client and server.

This "ciphertext-in-ciphertext" is indistinguishable from random to everyone except the client and station. The 144-byte payload is encrypted using a strong symmetric block cipher (AES-128) in CBC mode, whose key is derived from the client-station shared secret. The remainder of the tag is the client's ECDH public point, encoded using Elligator 2 [23] over Curve25519 [22]. The encoded point is indistinguishable from uniform random due to the properties of the Elligator 2 encoding function.

Once the station has determined the connection is a tagged flow, it sets up a socket in the kernel to allow it to spoof packets from and receive packets for the server using the forge_socket kernel module. The station makes this socket non-blocking, and attaches an SSL object initialized with the extracted key block to it. The station then sends a response to the client over this channel, containing a confirmation that the station has picked up, and the number of bytes that the client is allowed to send toward this station before it must create a new connection.

5.7.3 Connection Limits

Because the server's connection with the client remains open, the server receives packets from the client, including data and acknowledgments for the station's data. The server will initially ignore these messages, however there are two instances where the server will send data. When it does so, the censor would be able to see this anomalous behavior, because the server will send data with stale sequence numbers and different payloads from what the station sent.

The first instance of the server sending data is when the server times out the connection at the application level. For example, web servers can be configured to timeout incomplete requests after a certain time, by using the mod_reqtimeout[3] module in Apache. We found through our development and testing the shortest timeout was 20 seconds, although most servers had much longer timeouts. We measured TLS hosts to determine how long they would take to time out or respond to an incomplete request similar to one used in TapDance. We measured a 1% sample of the IPv4 address space listening on port 443, and the Alexa

[3]http://httpd.apache.org/docs/2.2/mod/mod_reqtimeout.html

top million domains using ZMap [46], and found that many servers had timeouts longer than 5 minutes. Figure 5.5 shows the fraction of server timeouts.

The second reason a server will send observable packets back to the client is if the client sends it a sequence number that is outside of the server's current TCP receive window. This happens when the client has sent more than a window's worth of data to the station, at which point the server will respond with a TCP ACK packet containing the server's stale sequence and acknowledgment numbers, alerting an observant censor to the anomaly.

To prevent both of these instances from occurring in our implementation, we limit the connection duration to less than the server's timeout, and we limit the number of bytes that a client can send to the station to up to the server's receive window size. Receive window sizes after the TLS handshake completes are typically above about 16 KB. We note that the station is not limited to the number of bytes it can send to the client per connection, making the 16 KB limit have minimal impact on most download-heavy connections.

In the event that the client wants to maintain its connection for longer than the duration or send more than 16 KB, the client can reuse the 16-byte connection ID in a new E2M TLS connection to the server. The station will decode the connection ID and reconnect the new flow to the old proxy connection seamlessly. This allows the browser to communicate to the HTTP proxy indefinitely, without having to deal with the limitations of the underlying decoy connection.

5.8 Evaluation

Throughout our evaluation, we used a client running Ubuntu 13.10 connected to a university network over gigabit Ethernet. For our decoy server, we used a Tor exit server at our institution, with a gigabit upstream through an HP 6600-24G-4XG switch. For our ISP station, we used a 16-core Supermicro server with 64 GB of RAM, connected via gigabit NICs to an upstream and to a mirror port from the HP switch. Our ISP station is therefore able to observe (but not block) packets to the Tor exit server, which provides a reasonable amount of background traffic on the order of 160 Mbps. In our tests, the Tor exit node generates a modest amount of realistic user traffic. Although not anywhere near the bandwidth of a Tier-1 ISP, Tor exit nodes generate a greater ratio of HTTPS flows than a typical ISP (due to the Tor browser's inclusion of the HTTPS Everywhere plugin), and we can use this microbenchmark to perform a back-of-the-envelope calculation to the loads we would see at a 40 Gbps Transit ISP tap.

We evaluate our proof-of-concept implementation with the goal of demonstrating that our system operates as described, and that our implementation is able to function within the

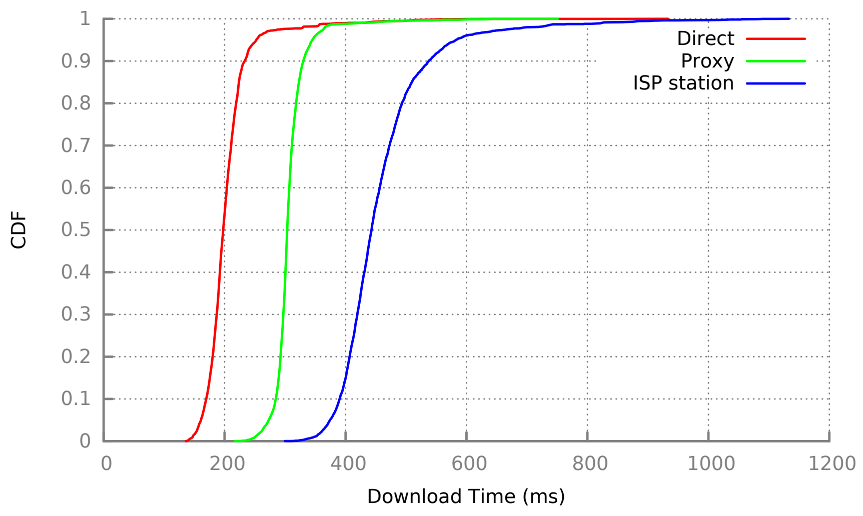


Figure 5.4: **Download Times Through TapDance** — We used Apache Benchmark to download www.facebook.com 5000 times (with a concurrency of 100) over normal HTTPS, through a single-hop proxy, and through our TapDance proof-of-concept.

constraints of our mock-ISP. To demonstrate that our system operates as described, we set Firefox to use our client as a proxy, and browsed several websites while capturing packets on the client and the decoy server. We then manually inspected the recorded packets to confirm that there were no additional packets sent by the server that would reveal our connections to be proxied connections. Empirically, we note that we are able to easily browse the Internet through this proxy, for example watching high-definition YouTube videos.

To evaluate the performance of our system, we created 8 proxy processes on our ISP station, using the same PF_RING cluster ID in order to share the load across 8 cores. The background traffic from the Tor exit server does not appear to have a significant impact on the proxy's load: each process handles between 20 and 50 flows at a given time, comprising up to 35 Mbps of TLS traffic. The CPU load during this time was less than 1%.

We used Apache Benchmark[4] in order to issue 5,000 requests through our station proxy, with a concurrency of 100, and compared the performance for fetching a simple page over HTTP and over HTTPS. We also compare fetching the same pages directly from the server and through a single-hop proxy. Figure 5.4 shows the cumulative distribution function for the total time to download the page. Although there is a modest overhead for end-to-middle proxy connections compared to direct or simple proxies, the overhead is not prohibitive to web browsing habits; users are still able to interact with the page, and pages can be expected to load in a reasonable time period. In particular, our proxy adds a median latency of 270 milliseconds to a page download in our tests when compared with a direct download.

[4]http://httpd.apache.org/docs/2.2/programs/ab.html

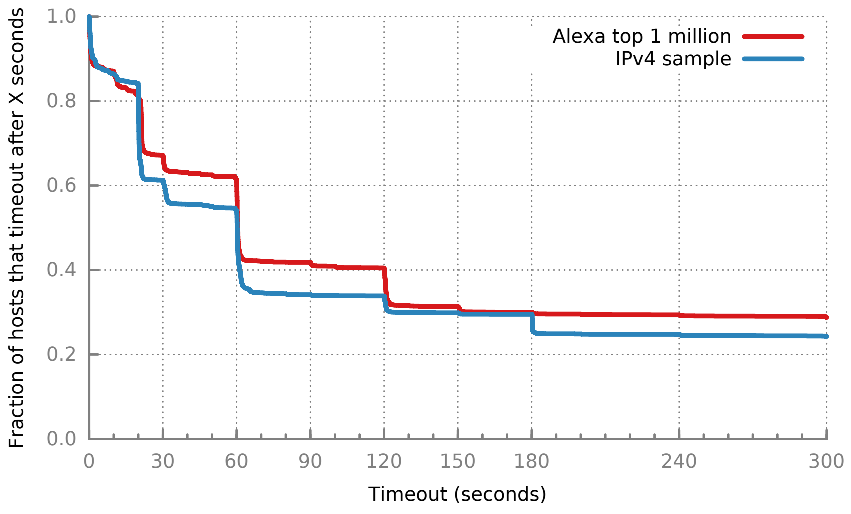


Figure 5.5: **Timeouts for Decoy Destinations** — To measure how long real TLS hosts will leave a connection open after receiving the incomplete request used in TapDance, we connected to two sets of TLS hosts (the Alexa top 1 million and a 1% sample of the IPv4 address space). We sent TapDance's incomplete request and timed how long the host would leave the connection open before either sending data or closing the connection. We find that over half the hosts will allow connections 60 seconds or longer.

We find that the CPU performance is bottlenecked by our single-threaded client. During our tests, the client consumes 100% CPU on a single core, while each of the 8 processes on the ISP station consume between 4-7% CPU. We also observe that a majority of the download time is spent waiting for the connection handshake to complete with the server. To improve this performance, we could speculatively maintain a connection pool in order to decrease the wait-time between requests. However, care must be taken in order to mimic the same connection pool behaviors that a browser might exhibit.

We also note that although the distribution of download times appear different for ISP station vs. normal connections, this does not necessarily indicate an observable feature for a censor. This is because our download involves a second round trip between client and server before the data reaches the client. The censor would still have to distinguish between this type of connection behavior and innocuous HTTP pipelined connections. It still may be possible for the censor to distinguish, however, as we discussed in Section 5.5, traffic analysis is an open problem for existing network proxies, and outside the scope of TapDance.

Tag creation and processing In order to evaluate the overhead of creating and checking for tags, we timed the creation and evaluation of 10,000 tags. We were able to create over 2,400 tags/second on our client and verify over 12,000 tags/second on a single core of our

ISP station. We find that the majority of time (approximately 80%) during tag creation is spent performing the expected three ECC point multiplications (an expected two to generate the client's Elligator-encodable public point and one to generate the shared secret). Similarly, during tag checking, nearly 90% of the computation time is spent on the single ECC point multiplication. Faster ECC implementations (such as tuned-assembly or ASICs) could have a significant impact toward improving the performance of tag verification on the ISP station.

Server support In order to measure how many servers can act as decoy destinations, we probed samples of the IPv4 address space as well as the Alexa top million hosts with tests to indicate support for TapDance. In our first experiment, we tested how long servers would wait to timeout an incomplete request, such as the one used by the client in TapDance. We scanned TLS servers in a 1% sample of the IPv4 address space, as well as the Alexa top million hosts, and sent listening servers a TLS handshake, followed by an incomplete HTTP request containing the full range of characters used in the TapDance client. We timed how long each server waited to either respond or close the connection. Servers that responded immediately do not support the TapDance incomplete request, either because they do not support incomplete requests, or the request contained characters outside the allowed range. Figure 5.5 shows the results of this experiment. For the 20-second timeout used in our implementation, over 80% of servers supported our incomplete request.

We also measured how servers handled the out-of-sequence TCP packets sent by the TapDance client, including packets acknowledging data not yet sent by the server. Again, we used a 1% sample of the IPv4 address space and the Alexa top million hosts. For each host, we connected to port 80 and sent an incomplete HTTP request, followed by a TCP ACK packet and a data-containing packet, both with acknowledgements set 100 bytes ahead of the true value. We find that the majority of Alexa servers still allow such packets, however, older or embedded systems often respond to our probes, in violation of the TCP specification. We conclude that TapDance clients must carefully select which servers they use as end points, but that there is no shortage of candidates from which to select.

5.9 Future Work

The long-term goal of end-to-middle proxies is to be implemented and deployed in a way that effectively combats censorship. While we have suggested a design that we believe is more feasible than previous work, more engineering must be done to bring it to maturity.

For example, deploying an end-to-middle proxy such as TapDance at an ISP requires not only scaling up to meet the demands of proxy users, but also of the deploying ISP's non-proxy traffic, which can be on the order of gigabits per second. One potential solution to this problem is to make the ISP component as stateless as possible. Extending TapDance, it may be possible to construct a "single-packet" version of an end-to-middle proxy. In this version the client uses the ciphertext steganographic channel to encode its entire request to the proxy. The proxy needs only detect these packets, fetch the requesting page, and inject a response. Such a design would not need to reconstruct TCP flows or keep state across multiple packets, allowing it to handle higher bandwidths of traffic, at the expense of making active attacks easier to perform by an adversary. Further investigation may discover an optimal balance between these tradeoffs.

Another open research question is where specifically in the network such proxies should be deployed. Previously, "Routing around Decoys" [130] outlined several novel attacks that a censor could perform in order to circumvent many anticensorship deployment strategies. There is ongoing discussion in the literature about the practical costs of these attacks, and practical countermeasures deployments could take to protect against them [70, 31].

As mentioned in Section 5.5, traffic fingerprinting is a concern for all proxies, and remains an open problem. Previous work has discussed these attacks as they apply to ISP-located proxies [130] and other covert channel proxies [67, 59]. Future work in this direction could provide insight into how to generate or mimic network traffic and protocols.

Finally, there is room to explore more active defense techniques, as outlined in Section 5.5. As end-to-middle proxies become more prominent, this is likely to become an important problem, as China has already started to employ active attacks in order to detect and censor Tor bridge relays [42, 167, 168]. Collaborating with ISPs will allow us to explore the technical capabilities and policies that would permit active defense against these attacks.

5.10 Related Work

Related steganographic techniques Other techniques [104, 13, 20] leverage pseudorandom public-key encryption (i.e., encryption that produces ciphertext indistinguishable from random bits) in order to solve the classic prisoners' problem. These techniques allow protocol participants to produce messages that mimic the distribution of an "innocent-looking" communication channel. The problem setting differs from ours, however, and the encoding of hidden messages inside an allowed encrypted channel (as valid ciphertexts) is not considered.

Dyer et al. [47] introduce a related technique called format transforming encryption (FTE), which disguises encrypted application-layer traffic to look like an innocent, allowed protocol from the perspective of deep packet inspection (DPI) technologies. The basic notion is to transform ciphertexts to match an expected format; as DPI technologies typically use membership in a regular language to classify application-layer traffic, FTE works by using a (bijective) encoding function that maps a ciphertext to a member of a pre-specified language. This steganographic technique differs significantly from ours, in that we do not attempt to disguise the use of a particular internet protocol itself (i.e., TLS), but rather ensure that our encoded ciphertext does not alter the expected distribution of the selected protocol traffic (i.e., TLS ciphertexts, in our system design).

5.11 Conclusion

End-to-middle proxies are a promising concept that may help tilt the balance of power from censors to citizens. Although previous designs including Telex, Cirripede, and Decoy Routing have laid the ground for this new direction, there are several problems when it comes to deploying any of these designs in practice. Previous designs have required inline blocking elements and sometimes assumed symmetric network paths. To address these concerns, we have developed TapDance, a novel end-to-middle proxy that operates without the need for inline flow blocking. We also described a novel way to support asymmetric flows without inline-flow blocking, by encoding arbitrary-length steganographic payloads in ciphertext. This covert channel may be independently useful for future E2M schemes and other censorship resistance applications.

By further studying how to make E2M schemes practical, we learn how code-as-law systems fail in practice. Specifically, systems like TapDance show how it is possible to engineer around practical concerns of deploying first-generation E2M proxies like Telex, without sacrificing major functionality. If extended to previous chapters, this lesson could be used to affirm previous test-environment security studies' results: vulnerabilities in code-as-law systems can be exploited in practice, and present a clear danger in deploying them, even if the studies revealing these vulnerabilities do not exploit them in the field.

Ultimately, anticensorship proxies are only useful if they are actually deployed. We hope that removing these barriers to end-to-middle proxying is a step towards that goal, allowing Internet freedom-supporting governments and organizations to leverage new policy and resource incentives to encourage such proxies to be deployed.

CHAPTER 6

Conclusion

The world is heading toward a more automated future. As Marc Andreesen writes, "software is eating the world" [17], meaning that software is coming more and more into our everyday lives, taking over the way we communicate, do business, and live. However, we must take caution of what and how we implement critical infrastructure, such as implementing law and public policy as code. This dissertation investigates three instances of software-only implementations of policy and where they go wrong:

1. Internet voting is susceptible to attack by exploiting the centralized servers that run the election, or even the computers that voter's cast their ballots. In this case, the code that runs the election does not know the intent of the election to fairly count the votes.

2. In the case of airport security, attackers can conceal contraband past a checkpoint by tricking the method the backscatter X-ray machines use to find plastic explosives. Because these machines do not understand their intent, simply masking a shadow is enough to fool them.

3. Finally, Internet censors that implement checks for banned websites in software can be tricked to allow traffic through that appears to be legitimate, but is instead accessing a proxy that the censor wishes to ban. Since network packets do not convey the intent or their true destination, firewalls cannot easily tell what information or action a packet will induce.

In each case, computers lack the ability to understand the intended behavior they have been coded to perform. Whereas human judges and officers can discern this, computers are left to blindly follow their instructions literally.

Contributing to the brittleness of this process, the decisions made by computers are often irreversible. For example, in Internet voting, once ballots have been committed to server hard drives and votes tabulated, there is no reversing the outcome. It is not possible to tell if the votes written on the hard drive match those that were cast by the voters or if they have been overwritten by an attacker. Similarly in airport security, after a decision is made on whether a passenger is concealing contraband, the passenger is allowed to board the plane, and reversing this decision at a later time is difficult at best. In Internet censorship, the decision to drop or forward packets must be made in mere milliseconds. With or without the ability to rollback each decision, we place our fate in the handles of computers. Actions made by computers, whether mistaken or unerring, are difficult or impossible to undo or appeal without external evidence or independent logs.

One way to combat the irreversible nature of computer's actions is to build redundant logging into systems in ways that cannot be tampered with by attackers. For example, votes in online elections could both be recorded by the server and verified and stored on citizens' computers. This makes it more difficult for an attacker who compromises only the server to change the outcome of the election. As another example, logs could be stored in append-only data structures [119] so that an attacker cannot tamper with events recorded prior to the attacker's infiltration.

Another idea is to use a hybrid human-computer approach. However, it can be difficult to design a hybrid system so that the strengths (rather than the weaknesses) of both are compounded. As an example, self-driving cars have a challenge of deciding how and when to yield control of the vehicle back to the human [142]. This is challenging both because the software has to recognize when it is in a situations it is specifically not good at navigating or reasoning about, and because it has to recognize these situations far enough in advance for the human driver to have time to safely take over. For some scenarios—such as inclement weather—this is relatively easy to predict. But for others, such as an unknown object that appears to be heading into the path of the vehicle, the software may not know whether to apply brakes, alert the driver, or continue on. By altering the course of the vehicle, the software may actually increase the risk of an accident if the object turns out to be benign (such as a paper bag) and there are other real hazards around.

Despite these challenges, hybrid approaches that successfully combine human and computer strengths are possible. One example is in weather forecasting, where modern computer models try to predict the weather through simulating with initial conditions of the current weather. Because the system model is chaotic, even small changes in the initial conditions can lead to drastic changes in forecast. Computers can help this by running large numbers of simulations, with slight variations in the initial conditions, and aggregating the results into a single forecast [133]. However, these computer models can often miss important patterns in their forecasts, and allowing humans to inspect and alter what simulations and initial conditions are run can improve the accuracy of temperature forecasts by about 10% [133, 73]. In this case, the human forecasters are aiding computers by visibly seeing patterns that are difficult for computers to recognize [133]. By recognizing and addressing the gaps in software implementations of weather forecasting, predictions can be improved beyond what either humans or computers can produce on their own.

Lessons One lesson from these case studies is that mechanistic evaluation of rules by computers is brittle, especially in the presence of adversaries. In public policy and law, adversaries are common (otherwise, there would be no need for a policy or law to be enforced in the first place), yet computer programs fail to fully capture the intricate semantics and intent behind the policies and law, leaving them vulnerable to circumvention. To combat this, humans should be included in a way that allows them to detect when enforcement does not match the intended behavior of a program, either through audit logs, additional checks, or random verification.

Another lesson is the risks we face when computers are allowed to make irreversible decisions. Computers make mistakes and it is important to be able to undo them after closer inspection. Absent this, it is the attackers and criminals who have the final say in our vulnerable society.

Moving forward, automated systems must at least allow themselves to be checked, either by keeping append-only audit records, or by allowing inspection by humans during operation. This will allow humans and computers to both contribute to the security and efficiency of previously-expensive tasks. While potentially more costly, this hybrid approach will strengthen the security of these systems until software development is able to easily capture the intent of the developer.

BIBLIOGRAPHY

[1] vol. 74 U.S. 482. 1868.

[2] Driverless tractor plants crops in spirals. *Popular Mechanics* (July 1940), 7.

[3] vol. 508 U.S. 223. 1993.

[4] vol. 552 U.S. 74. 2007.

[5] Internet voting in Estonia. Vabariigi Valimiskomisjon. http://www.vvk.ee/public/dok/Internet_Voting_in_Estonia.pdf, Feb. 2007.

[6] Uncovering the veil on Geneva's Internet voting solution. Republique Et Canton De Geneve http://www.geneve.ch/evoting/english/doc/Flash_IT_vote_electronique_SIDP_final_english.pdf, Feb. 2009.

[7] District of Columbia's Board of Elections and Ethics adopts open source digital voting foundation technology to support ballot delivery. OSDV Press Release. http://osdv.org/wp-content/uploads/2010/06/osdv-press-release-final-62210.pdf, June 2010.

[8] Internet voting, still in beta. The New York Times editorial. http://www.nytimes.com/2010/01/28/opinion/28thu4.html, Jan. 2010.

[9] Internet voting. Verified Voting. http://www.verifiedvoting.org/article.php?list=type&type=27, May 2011.

[10] ab: Apache benchmark. http://httpd.apache.org/docs/2.0/programipsets/ab.html.

[11] ADIDA, B. Helios: Web-based open-audit voting. In *Proc. 17th USENIX Security Symposium* (July 2008).

[12] AHLERS, M. M. TSA removing "virtual strip search" body scanners. CNN, Jan. 2013. http://www.cnn.com/2013/01/18/travel/tsa-body-scanners.

[13] AHN, L., AND HOPPER, N. J. Public-key steganography. In *EUROCRYPT 2004*, vol. 3027 of *LNCS*. Springer Berlin Heidelberg, 2004, pp. 323–341.

[14] Alexa top sites. http://www.alexa.com/topsites.

[15] American Association of Physicists in Medicine. Radiation dose from airport scanners. Tech. Rep. 217, June 2013. http://www.aapm.org/pubs/reports/RPT_217.pdf.

[16] Anderson, R. J., and Petitcolas, F. A. P. On the limits of steganography. *IEEE J. Sel. A. Commun. 16*, 4 (Sept. 2006), 474–481.

[17] Andreessen, M. Why software is eating the world. http://www.wsj.com/articles/SB10001424053111903480904576512250915629460, Aug. 2011.

[18] Appel, A. W., Ginsburg, M., Hursti, H., Kernighan, B. W., Richards, C. D., Tan, G., and Venetis, P. The New Jersey voting-machine lawsuit and the AVC Advantage DRE voting machine. In *Proc. 2009 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE)* (Aug. 2009).

[19] Aryan, S., Aryan, H., and Halderman, J. A. Internet censorship in iran: A first look. *Free and Open Communications on the Internet, Washington, DC, USA* (2013).

[20] Backes, M., and Cachin, C. Public-key steganography with active attacks. In *Theory of Cryptography Conference – TCC '05* (2005), vol. 3378 of *LNCS*, Springer Berlin Heidelberg, pp. 210–226.

[21] Bates, M., and Weischedel, R. M. *Challenges in natural language processing*. Cambridge University Press, 2006.

[22] Bernstein, D. J. Curve25519: New Diffie-Hellman speed records. In *Public Key Cryptography – PKC 2006*, vol. 3958 of *LNCS*. Springer Berlin Heidelberg, 2006, pp. 207–228.

[23] Bernstein, D. J., Hamburg, M., Krasnova, A., and Lange, T. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *ACM Conference on Computer and Communications Security – CCS 2013* (2013), ACM, pp. 967–980.

[24] Bloomberg. How a mystery trader with an algorithm may have caused the flash crash. http://www.bloomberg.com/news/articles/2015-04-22/mystery-trader-armed-with-algorithms-rewrites-flash-crash-story, April 2015.

[25] Bos, J. W., Kaihara, M. E., Kleinjung, T., Lenstra, A. K., and Montgomery, P. L. Playstation 3 computing breaks 2^{60} barrier 112-bit prime ECDLP solved. http://lacal.epfl.ch/112bit_prime, 2009.

[26] Bowen, D., et al. "Top-to-Bottom" Review of voting machines certified for use in California. Tech. rep., California Secretary of State, 2007. http://sos.ca.gov/elections/elections.vsr.htm.

[27] BROGAARD, J. High frequency trading and its impact on market quality. *Northwestern University Kellogg School of Management Working Paper* (2010), 66.

[28] BURNETT, S., FEAMSTER, N., AND VEMPALA, S. Chipping away at censorship firewalls with user-generated content. In *19th USENIX Security Symposium* (2010), USENIX Association, pp. 463–468.

[29] BUTLER, K., ENCK, W., HURSTI, H., MCLAUGHLIN, S., TRAYNOR, P., AND MCDANIEL, P. Systemic issues in the Hart InterCivic and Premier voting systems: Reflections on project EVEREST. In *Proc. 2008 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE)* (July 2008).

[30] CERRA, F. Assessment of the Rapiscan Secure 1000 body scanner for conformance with radiological safety standards, July 2006. http://www.tsa.gov/sites/default/files/assets/pdf/research/rapiscan_secure_1000.pdf.

[31] CESAREO, J., KARLIN, J., REXFORD, J., AND SCHAPIRA, M. Optimizing the placement of implicit proxies. http://www.cs.princeton.edu/~jrex/papers/decoy-routing.pdf, June 2012.

[32] CNN. Shoe bomb suspect to remain in custody. CNN, Dec. 2001. http://edition.cnn.com/2001/US/12/24/investigation.plane/.

[33] COMPTON, A. H. A quantum theory of the scattering of X-rays by light elements. *Physical Review 21*, 5 (1923), 483.

[34] CORBETT, J. $1B of TSA nude body scanners made worthless by blog: How anyone can get anything past the scanners, Mar. 2012. http://tsaoutofourpants.wordpress.com/2012/03/06/1b-of-nude-body-scanners-made-worthless-by-blog-how-anyone-can-get-anything-past-the-tsas-nude-body-scanners.

[35] DANEZIS, G., AND ANDERSON, R. The economics of censorship resistance. In *Proceedings of the 3rd Annual Workship on Economics and Information Security (WEIS04)* (May 2004).

[36] DANEZIS, G., AND DIAZ, C. Survey of anonymous communication channels. *Computer Communications 33* (Mar. 2010).

[37] DEPARTMENT OF HOMELAND SECURITY, OFFICE OF INSPECTOR GENERAL. Transportation Security Administration's use of backscatter units. Tech. Rep. OIG-12-38, Feb. 2012. http://www.oig.dhs.gov/assets/Mgmt/2012/OIG_12-38_Feb12.pdf.

[38] DEPARTMENT OF HOMELAND SECURITY, SCIENCE AND TECHNOLOGY DIRECTORATE. Compilation of emission safety reports on the L3 Communications, Inc. ProVision 100 active millimeter wave advanced imaging technology (AIT) system.

Tech. Rep. DHS/ST/TSL-12/118, Sept. 2012. http://epic.org/foia/dhs/bodyscanner/appeal/Emission-Safety-Reports.pdf.

[39] DIERKS, T., AND RESCORLA, E. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878.

[40] DINGLEDINE, R. Strategies for getting more bridge addresses. https://blog.torproject.org/blog/strategies-getting-more-bridge-addresses, May 2011.

[41] DINGLEDINE, R. Obfsproxy: the next step in the censorship arms race. *Tor Project official blog* (2012). https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race.

[42] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium* (Aug. 2004).

[43] DIVISION, U. S. D. C. N. D. O. I. E. United states of america v. navinder singh sarao. http://www.justice.gov/sites/default/files/opa/press-releases/attachments/2015/04/21/sarao_criminal_complaint.pdf, April 2015.

[44] DOUCEUR, J. R. The Sybil attack. In *Proc. International Workshop on Peer-to-Peer Systems (IPTPS)* (2002), pp. 251–260.

[45] DURUMERIC, Z., KASTEN, J., BAILEY, M., AND HALDERMAN, J. A. Analysis of the HTTPS certificate ecosystem. In *Internet Measurement Conference – IMC '13* (2013), ACM, pp. 291–304.

[46] DURUMERIC, Z., WUSTROW, E., AND HALDERMAN, J. A. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium* (Aug. 2013), USENIX Association, pp. 605–619.

[47] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 61–72.

[48] Dynaweb proxy. http://www.dit-inc.us/dynaweb.

[49] ENSAFI, R., KNOCKEL, J., ALEXANDER, G., AND CRANDALL, J. R. Detecting intentional packet drops on the internet via tcp/ip side channels. In *Passive and Active Measurement* (2014), Springer, pp. 109–118.

[50] EPIC. Transportation agency's plan to x-ray travelers should be stripped of funding, June 2005. http://epic.org/privacy/surveillance/spotlight/0605/.

[51] Esteghari, S., and Desmedt, Y. Exploiting the client vulnerabilities in Internet e-voting systems: Hacking Helios 2.0 as an example. In *Proc. 2010 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections (EVT/WOTE)* (Aug. 2010).

[52] Feamster, N., Balazinska, M., Harfst, G., Balakrishnan, H., and Karger, D. Infranet: Circumventing web censorship and surveillance. In *Proceedings of the 11th USENIX Security Symposium* (Aug. 2002).

[53] Feamster, N., Balazinska, M., Wang, W., Balakrishnan, H., and Karger, D. Thwarting web censorship with untrusted messenger discovery. In *Privacy Enhancing Technologies* (2003), Springer, pp. 125–140.

[54] Feldman, A. J., Halderman, J. A., and Felten, E. W. Security analysis of the Diebold AccuVote-TS voting machine. In *Proc. 2007 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE)* (Aug. 2007).

[55] Fifield, D., Hardison, N., Ellithorpe, J., Stark, E., Boneh, D., Dingledine, R., and Porras, P. Evading censorship with browser-based proxies. In *Privacy Enhancing Technologies – PETS 2012* (2012), vol. 7384 of *LNCS*, Springer Berlin Heidelberg, pp. 239–258.

[56] Fifield, D., Lan, C., Hynes, R., Wegmann, P., and Paxson, V. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 1–19.

[57] for Voting Integrity, N. C. Recommendations for an optical scan ballot. http://votingintegrity.org/docs/help/opticalpc-ep.pdf.

[58] Fu, K. Trustworthy medical device software. In *Public Health Effectiveness of the FDA 510(k) Clearance Process: Measuring Postmarket Performance and Other Select Topics: Workshop Report* (July 2011).

[59] Geddes, J., Schuchard, M., and Hopper, N. Cover your ACKs: Pitfalls of covert channel censorship circumvention. In *ACM Conference on Computer and Communications Security – CCS 2013* (2013), ACM, pp. 361–372.

[60] Global Internet Freedom Consortium. The Great Firewall revealed. http://www.internetfreedom.org/files/WhitePaper/ChinaGreatFirewallRevealed.pdf.

[61] Global Internet Freedom Consortium. Defeat internet censorship: Overview of advanced technologies and products. http://www.internetfreedom.org/archive/Defeat_Internet_Censorship_White_Paper.pdf, Nov. 2007.

[62] Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082* (2013).

[63] GOOGLE. Inside google translate. https://translate.google.com/about/intl/en_ALL/.

[64] HALDERMAN, J. A., AND TEAGUE, V. The new south wales ivote system: Security failures and verification flaws in a live online election. *arXiv preprint arXiv:1504.05646* (2015).

[65] HANDEL, T. G., AND SANDFORD, II, M. T. Hiding data in the OSI network model. In *Information Hiding – IH '96* (1996), vol. 1174 of *LNCS*, Springer Berlin Heidelberg, pp. 23–38.

[66] HINTZ, A. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, R. Dingledine and P. Syverson, Eds., vol. 2482 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 229–233.

[67] HOUMANSADR, A., BRUBAKER, C., AND SHMATIKOV, V. The parrot is dead: Observing unobservable network communications. In *IEEE Symposium on Security and Privacy – SP '13* (2013), IEEE, pp. 65–79.

[68] HOUMANSADR, A., NGUYEN, G. T. K., CAESAR, M., AND BORISOV, N. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *ACM Conference on Computer and Communications Security – CCS 2011* (2011), ACM, pp. 187–200.

[69] HOUMANSADR, A., RIEDL, T., BORISOV, N., AND SINGER, A. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *Network and Distributed System Security Symposium – NDSS 2013* (2013), Internet Society.

[70] HOUMANSADR, A., WONG, E. L., AND SHMATIKOV, V. No direction home: The true cost of routing around decoys. In *Network and Distributed System Security Symposium – NDSS '14* (2014), Internet Society.

[71] HUBBARD, J. New jail to have x-ray scanner used for security at airport. Wilkes Journal-Patriot, Oct. 2013. http://www.journalpatriot.com/news/article_95a398bc-368d-11e3-99ec-0019bb30f31a.html.

[72] HUGHES, R. Systems and methods for improving directed people screening, June 12 2012. US Patent 8,199,996.

[73] HYDRO METEOROLOGICAL PREDICITION CENTER, N. O., AND ASSOCIATE, A. Hpc pct improvements vs mos. http://www.wpc.ncep.noaa.gov/images/hpcvrf/max1.gif.

[74] INVERNIZZI, L., KRUEGEL, C., AND VIGNA, G. Message in a bottle: Sailing past censorship. In *29th Annual Computer Security Applications Conference – ACSAC 2013* (2013), ACM, pp. 39–48.

[75] IP sets. http://ipset.netfilter.org/.

[76] IVW-AGNE. Rapiscan Secure 1000 DP (Dual Pose) backscatter body scanner / nacktscanner. eBay listing, 2012. http://www.ebay.com/itm/Rapiscan-Secure-1000-DP-Dual-Pose-Backscatter-Body-Scanner-Nacktscanner-/110999548627.

[77] JEFFERSON, D., RUBIN, A. D., SIMONS, B., AND WAGNER, D. A security analysis of the secure electronic registration and voting experiment (SERVE). http://servesecurityreport.org/paper.pdf, Jan. 2004.

[78] JOHNS HOPKINS UNIVERSITY APPLIED PHYSICS LABORATORY. Radiation safety engineering assessment report for the Rapiscan Secure 1000 in single pose configuration. Tech. Rep. NSTD-09-1085, version 2, Aug. 2010. http://www.tsa.gov/sites/default/files/assets/pdf/research/jh_apl_v2.pdf.

[79] JOHNSTON, R. G. Adversarial safety analysis: Borrowing the methods of security vulnerability assessments. *J. Safety Research 35*, 3 (2004), 245–48.

[80] JUELS, A., AND BRAINARD, J. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the 1999 Network and Distributed System Security Symposium (NDSS)* (Feb. 1999).

[81] KARLIN, J., ELLARD, D., JACKSON, A. W., JONES, C. E., LAUER, G., MANKINS, D. P., AND STRAYER, W. T. Decoy routing: Toward unblockable Internet communication. In *USENIX Workshop on Free and Open Communications on the Internet – FOCI '11* (2011), USENIX Association.

[82] KASTEN, J., WUSTROW, E., AND HALDERMAN, J. A. CAge: Taming certificate authorities by inferring restricted scopes. In *Financial Cryptography and Data Security – FC 2013*, vol. 7859 of *LNCS*. Springer Berlin Heidelberg, 2013, pp. 329–337.

[83] KAUFMAN, L., AND CARLSON, J. W. An evaluation of airport X-ray backscatter units based on image characteristics. *Journal of Transportation Security 4*, 1 (2011), 73–94.

[84] KENNA, B., AND MURRAY, D. Evaluation tests of the SECURE 1000 scanning system. Tech. Rep. SAND 91-2488, UC-830, Sandia National Laboratories, Apr. 1992.

[85] KIAYIAS, A., KORMAN, M., AND WALLUCK, D. An Internet voting system supporting user privacy. In *22nd Annual Computer Security Applications Conference* (Los Alamitos, CA, USA), IEEE Computer Society, pp. 165–174.

[86] KOHNO, T., STUBBLEFIELD, A., RUBIN, A. D., AND WALLACH, D. S. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy* (May 2004), pp. 27–40.

[87] KOSCHER, K., CZESKIS, A., ROESNER, F., PATEL, S., KOHNO, T., CHECKOWAY, S., MCCOY, D., KANTOR, B., ANDERSON, D., SHACHAM, H., AND SAVAGE, S. Experimental security analysis of a modern automobile. In *Proc. 31st IEEE Symposium on Security and Privacy* (May 2010), pp. 447–62.

[88] KOTOWSKI, A., AND SMITH, S. X-ray imaging system with active detector, Dec. 16 2003. US Patent 6,665,373.

[89] KRAVETS, D. Court oks airport body scanners, rejects constitutional challenge, July 2011. http://www.wired.com/threatlevel/2011/07/court-approves-body-scanners/.

[90] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[91] LANGLEY, A. TLS symmetric crypto. https://www.imperialviolet.org/2014/02/27/tlssymmetriccrypto.html, Feb. 2014.

[92] LANGNER, R. To kill a centrifuge: A technical analysis of what stuxnet's creators tried to achieve. Online: http://www.langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf, Nov. 2013.

[93] LESSIG, L. *Code and other laws of cyberspace*. 1999.

[94] LESSIG, L. *Code: Version 2.0*. 2006.

[95] LEWIS, M. *Flash Boys: A Wall Street Revolt*. 2014.

[96] LIMITED, S. E. Shazam: Music discovery, charts & song lyrics. http://www.shazam.com/.

[97] LOWREY, A. My visit to the offices of Rapiscan, which makes airport scanners. Slate, Nov. 2010. http://www.slate.com/articles/business/moneybox/2010/11/corporate_junket.html.

[98] LYON, G. *Nmap Network Scanning*. Nmap Project, 2009, ch. Chapter 8: Remote OS Detection.

[99] MANNING, J. F. The absurdity doctrine. *Harvard Law Review* (2003), 2387–2486.

[100] MATHEWSON, N., AND PROVOS, N. libevent: An event notification library. http://libevent.org/.

[101] MCCROHAN, J. L., AND SHELTON WATERS, K. R. Response to UCSF regarding their letter of concern [131], Oct. 2010. http://www.fda.gov/Radiation-EmittingProducts/RadiationEmittingProductsandProcedures/SecuritySystems/ucm231857.htm.

[102] Medici, A. Guess where TSA's invasive scanners are now? Federal Times, May 2014. http://www.federaltimes.com/article/20140516/DHS/305160012/Guess-where-TSA-s-invasive-scanners-now-.

[103] Mohajeri Moghaddam, H., Li, B., Derakhshani, M., and Goldberg, I. SkypeMorph: Protocol obfuscation for Tor bridges. In *ACM Conference on Computer and Communications Security – CCS 2012* (2012), ACM, pp. 97–108.

[104] Möller, B. A public-key encryption scheme with pseudo-random ciphertexts. In *Computer Security – ESORICS 2004*, vol. 3193 of *LNCS*. Springer Berlin Heidelberg, 2004, pp. 335–351.

[105] Mowery, K., Wustrow, E., Wypych, T., Singleton, C., Comfort, C., Rescorla, E., Checkoway, S., Halderman, J. A., and Shacham, H. Security analysis of a full-body scanner. In *Proceedings of the 23rd USENIX Security Symposium* (Aug. 2014), K. Fu, Ed., USENIX.

[106] Murdoch, S. J., Bond, M., and Anderson, R. How certification systems fail: Lessons from the Ware report. *IEEE Security & Privacy 10*, 6 (Nov–Dec 2012), 40–44.

[107] Murdoch, S. J., and Lewis, S. Embedding covert channels into TCP/IP. In *Information Hiding – IH '05*, vol. 3727 of *LNCS*. Springer Berlin Heidelberg, 2005, pp. 247–261.

[108] National Security Agency. Cottonmouth-iii. *Der Spiegel* (2013). http://www.spiegel.de/international/world/a-941262.html, fetched 2014-02-27.

[109] Newth, M. The long history of censorship. http://www.beaconforfreedom.org/liste.html?tid=415&art_id=475, 2010.

[110] ntop.org. PF_RING: High-speed packet capture, filtering and analysis. http://www.ntop.org/products/pf_ring/.

[111] OSI Systems. OSI Systems receives $25m order from U.S. Transportation Security Administration for advanced imaging technology. Press release, Oct. 2009. http://investors.osi-systems.com/releasedetail.cfm?ReleaseID=413032.

[112] Paxson, V. Bro: A system for detecting network intruders in real time. *Computer Networks 31* (1999), 2435–2463.

[113] Peterson, L., Anderson, T., Culler, D., and Roscoe, T. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of HotNets–I* (Princeton, New Jersey, October 2002).

[114] Phantom Laboratory. The rando phantom, ran100 and ran110, 2006. http://www.phantomlab.com/library/pdf/rando_datasheet.pdf.

[115] PLUNGIS, J. Naked-image scanners to be removed from U.S. airports. Bloomberg News, Jan. 2013. http://www.bloomberg.com/news/2013-01-18/naked-image-scanners-to-be-removed-from-u-s-airports.html.

[116] POSTEL, J. Transmission Control Protocol. RFC 793 (Internet Standard), Sept. 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[117] PROJECT, O. OpenSSL: Cryptography and SSL/TLS toolkit. http://www.openssl.org/.

[118] PROJECT, T. Tor: Pluggable transports. https://www.torproject.org/docs/pluggable-transports.html.en.

[119] PULLS, T., AND PEETERS, R. Balloon: A forward-secure append-only persistent authenticated data structure. In *Computer Security – ESORICS 2015* (2015), vol. 9327 of *LNCS*.

[120] RAPISCAN SYSTEMS. Rapiscan secure 1000, 2005. http://epic.org/privacy/surveillance/spotlight/0605/rapiscan.pdf.

[121] RAPISCAN SYSTEMS. *Secure 1000 Personnel Scanner Operator's Manual*, Aug. 2005.

[122] RAPISCAN SYSTEMS. Rapiscan 522b, 2006. http://www.wired.com/images_blogs/threatlevel/2014/02/RapiScan-522B.pdf, fetched 2014-02-27.

[123] RAPISCAN SYSTEMS. Rapiscan Secure 1000 health and safety fact sheet, 2012. http://www.rapiscansystems.com/extranet/downloadFile/24_Rapiscan%20Secure%201000-Health%20and%20Safety-Fact%20Sheet.pdf.

[124] RING, S., DETRIXHE, J., AND VAUGHAN, L. The alleged flash-trading mastermind lived with his parents and couldn't drive.

[125] ROGAWAY, P. Evaluation of some blockcipher modes of operation. Tech. rep., Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan, Feb. 2011.

[126] ROKEY W. SULEMAN, I., MCGHIE, K. W., TOGO D. WEST, J., AND LOWERY, C. Making reform a reality: An after-action report on implementation of the Omnibus Election Reform Act. DCBOEE. http://www.dcboee.org/popup.asp?url=/pdf_files/nr_687.pdf, Feb. 2011.

[127] ROSSIDES, G. D. TSA Reply to Rep. Bennie G. Thompson, Feb. 2010. http://epic.org/privacy/airtravel/backscatter/TSA_Reply_House.pdf.

[128] RUBIN, A. Security considerations for remote electronic voting over the Internet. http://avirubin.com/e-voting.security.html.

[129] SALOWEY, J., CHOUDHURY, A., AND MCGREW, D. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288 (Proposed Standard), Aug. 2008.

[130] SCHUCHARD, M., GEDDES, J., THOMPSON, C., AND HOPPER, N. Routing around decoys. In *ACM Conference on Computer and Communications Security – CCS 2012* (2012), ACM, pp. 85–96.

[131] SEDAT, J., AGARD, D., SHUMAN, M., AND STROUD, R. UCSF letter of concern, Apr. 2010. http://www.npr.org/assets/news/2010/05/17/concern.pdf.

[132] SHAHID, A. Feds admit they stored body scanner images, despite TSA claim the images cannot be saved, Aug. 2010. http://www.nydailynews.com/news/national/feds-admit-stored-body-scanner-images-tsa-claim-images-saved-article-1.200279.

[133] SILVER, N. *The Signal and the Noise: Why So Many Predictions Fail-but Some Don't*. September 2012.

[134] SIMMONS, G. J. The prisoners' problem and the subliminal channel. In *CRYPTO '83*. Springer US, 1984, pp. 51–67.

[135] SMITH, S. W. Secure 1000, concealed weapon detection system, 1998. http://www.dspguide.com/secure.htm, fetched 2014-02-27.

[136] SMITH, S. W. Re: Misinformation on airport body scanner radiation safety, Dec. 2010. http://tek84.com/downloads/radiation-bodyscanner.pdf.

[137] SMITH, S. W. Resume, 2014. http://www.dspguide.com/resume.htm, fetched 2014-02-27.

[138] SOGHOIAN, C., AND STAMM, S. Certified lies: Detecting and defeating government interception attacks against SSL (short paper). In *Financial Cryptography and Data Security – FC 2011*, vol. 7035 of *LNCS*. Springer Berlin Heidelberg, 2012, pp. 250–259.

[139] SPRINGALL, D., FINKENAUER, T., DURUMERIC, Z., KITCAT, J., HURSTI, H., MACALPINE, M., AND HALDERMAN, J. A. Security analysis of the Estonian Internet voting system. In *Proceedings of the 21st ACM Conference on Computer and Communications Security* (Nov. 2014), ACM.

[140] Squid HTTP proxy. http://www.squid-cache.org/.

[141] STENBJORN, P. An overview and design rationale memo. DCBOEE. http://www.dcboee.us/dvm/DCdVBM-DesignRationale-v3.pdf, Sept. 2010.

[142] STEWART, J. What may be self-driving cars' biggest problem.

[143] SURDAN, H. Computable contracts. *UCDL Rev. 46* (2012), 629.

[144] SZEGEDY, C. Building a deeper understanding of images. http://googleresearch.blogspot.com/2014/09/building-deeper-understanding-of-images.html, Sept. 2014.

[145] TATE, C. Privacy impact assessment for the Secret Service use of advanced imaging technology, Dec. 2011. http://epic.org/foia/dhs/usss/Secret-Service-Docs-1.pdf.

[146] TAYLOR, M., SMITH, R., DOSSING, F., AND FRANICH, R. Robust calculation of effective atomic numbers: The Auto-Zeff software. *Medical Physics 39*, 4 (2012), 1769.

[147] THE TOR PROJECT. Tor: Bridges. https://www.torproject.org/docs/bridges.

[148] THE TOR PROJECT. New blocking activity from Iran. https://blog.torproject.org/blog/new-blocking-activity-iran, Jan. 2011.

[149] TIMBERG, C. These hackers warned the internet would become a security disaster. nobody listened.

[150] Transparent proxy support documentation. http://lxr.linux.no/#linux+v2.6.37/Documentation/networking/tproxy.txt, Jan. 2011.

[151] TRANSPORTATION SECURITY ADMINISTRATION. Contract with rapiscan security products, June 2007. IDV ID: HSTS04-07-D-DEP344, http://epic.org/open_gov/foia/TSA_Rapiscan_Contract.pdf.

[152] TRANSPORTATION SECURITY ADMINISTRATION. Passenger screening using advanced imaging technology: Notice of proposed rulemaking. *Federal Register 78*, 58 (Mar. 2013), 18287–302.

[153] TRANSPORTATION SECURITY ADMINISTRATION OFFICE OF SECURITY TECHNOLOGY. Procurement specification for whole body imager devices for checkpoint operations. TSA, Sept. 2008. http://epic.org/open_gov/foia/TSA_Procurement_Specs.pdf.

[154] TSA. AIT: Frequently Asked Questions, May 2013. http://www.tsa.gov/ait-frequently-asked-questions. Fetched May 17, 2013: https://web.archive.org/web/20130517152631/http://www.tsa.gov/ait-frequently-asked-questions.

[155] TSA PRESS OFFICE. TSA takes next steps to further enhance passenger privacy, July 2011. http://www.tsa.gov/press/releases/2011/07/20/tsa-takes-next-steps-further-enhance-passenger-privacy.

[156] TÜV SÜD AMERICA. EMC test report: Secure 1000 WBI, Feb. 2009. http://epic.org/privacy/body_scanners/EPIC_TSA_FOIA_Docs_09_09_11.pdf, pages 162–323.

[157] UltraSurf proxy. http://www.ultrareach.com/.

[158] U.S. DEPARTMENT OF HOMELAND SECURITY OFFICE OF HEALTH AFFAIRS. Fact sheet: Advanced imaging technology (ait) health & safety, 2010. http://www.oregon.gov/OBMI/docs/TSA-AIT_ScannerFactSheet.pdf.

[159] VALLENTIN, M., SOMMER, R., LEE, J., LERES, C., PAXON, V., AND TIERNEY, B. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *Proc. 10th International Conference on Recent Advances in Intrusion Detection (RAID '07)* (Sept. 2007), pp. 107–126.

[160] VOTING, V. Internet voting outside the united states. https://www.verifiedvoting.org/internet-voting-outside-the-united-states/.

[161] WAH, E., HURD, D. R., AND WELLMAN, M. P. Strategic market choice: Frequent call markets vs. continuous double auctions for fast and slow traders. AAAI.

[162] WAIN, C. Lessons from lockerbie. BBC, Dec. 1998. http://news.bbc.co.uk/2/hi/special_report/1998/12/98/lockerbie/235632.stm.

[163] WANG, Q., GONG, X., NGUYEN, G. T. K., HOUMANSADR, A., AND BORISOV, N. CensorSpoofer: Asymmetric communication using ip spoofing for censorship-resistant web browsing. In *ACM Conference on Computer and Communications Security – CCS 2012* (2012), ACM, pp. 121–132.

[164] WATERS, B., JUELS, A., HALDERMAN, J. A., AND FELTEN, E. W. New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS 2004)* (Oct. 2004).

[165] WEINBERG, Z., WANG, J., YEGNESWARAN, V., BRIESEMEISTER, L., CHEUNG, S., WANG, F., AND BONEH, D. StegoTorus: A camouflage proxy for the Tor anonymity system. In *ACM Conference on Computer and Communications Security – CCS 2012* (2012), ACM, pp. 109–120.

[166] WENDLANDT, D., ANDERSEN, D. G., AND PERRIG, A. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX Annual Technical Conference – ATC '08* (2008), USENIX Association, pp. 321–334.

[167] WILDE, T. Great Firewall Tor probing. https://gist.github.com/twilde/da3c7a9af01d74cd7de7, 2012.

[168] WINTER, P., AND LINKDSKOG, S. How the Great Firewall of China is blocking Tor. In *2nd USENIX Workshop on Free and Open Communications on the Internet – FOCI '12* (Berkeley, CA, 2012), USENIX.

[169] WINTER, P., PULLS, T., AND FUSS, J. Scramblesuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society* (2013), ACM, pp. 213–224.

[170] WOLCHOK, S., WUSTROW, E., HALDERMAN, J. A., PRASAD, H. K., KANKIPATI, A., SAKHAMURI, S. K., YAGATI, V., AND GONGGRIJP, R. Security analysis of India's electronic voting machines. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS)* (Oct. 2010).

[171] WOLCHOK, S., WUSTROW, E., ISABEL, D., AND HALDERMAN, J. A. Attacking the washington, dc internet voting system. In *Financial Cryptography and Data Security*. Springer, 2012, pp. 114–128.

[172] WOLCHOK, S., YAO., R., AND HALDERMAN, J. A. Analysis of the Green Dam Censorware System. *Computer Science and Engineering Division, University of Michigan 18* (2009).

[173] WOLFGANG, J., DUSI, M., AND CLAFFY, K. C. Estimating routing symmetry on single links by passive flow measurements. ACM, pp. 473–478.

[174] WUSTROW, E., SWANSON, C. M., AND HALDERMAN, J. A. Tapdance: End-to-middle anticensorship without flow blocking. In *Proceedings of 23rd USENIX Security Symposium* (2014).

[175] WUSTROW, E., WOLCHOK, S., GOLDBERG, I., AND HALDERMAN, J. A. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium* (Aug. 2011), USENIX Association, pp. 459–474.

[176] XM MATERIALS. Material safety data sheet XM-03-X (Comp C-4 explosive simulant), Oct. 1999. http://www.xm-materials.com/MSDS/xray_msds/msdsxm_03_x.pdf.

[177] XM MATERIALS. Material safety data sheet XM-04-X (Semtex explosive simulant), Oct. 1999. http://www.xm-materials.com/MSDS/xray_msds/msdsxm_04_x.pdf.

[178] XU, X., MAO, Z., AND HALDERMAN, J. Internet censorship in China: Where does the filtering occur? In *12th Passive and Active Measurement Conference – PAM 2011* (2011), vol. 6579 of *LNCS*, pp. 133–142.