CS 4530: Fundamentals of Software Engineering

Module 15: Software Engineering & Security

Jon Bell, Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

Learning Objectives for this Module

- By the end of this module, you should be able to:
  - Define key terms relating to software/system security
  - Describe some of the tradeoffs between security and other requirements in software engineering
  - Explain 5 common vulnerabilities in web applications and similar software systems, and describe some common mitigations for each of them.
  - Explain why software alone isn’t enough to assure security

Outline of this lecture

1. Definition of key vocabulary
2. Some common vulnerabilities, and possible mitigations
3. Getting security right is about people as well as software.

Security: Basic Vocabulary (1)

- Security is a set of non-functional requirements (sometimes called “CIA”):
- Confidentiality: is information disclosed to unauthorized individuals?
- Integrity: is code or data tampered with?
- Availability: is the system accessible and usable?

Security: Basic Vocabulary (2)

- Asset: something of value that is the subject of a security requirement
- Threat: potential event that could compromise a security requirement
- Security architecture: a set of mechanisms and policies that we build into our system to mitigate risks from threats

Security: Basic Vocabulary (3)

- Vulnerability: a characteristic or flaw in system design or implementation, or in the security procedures, that, if exploited, could result in a security compromise
- Exploit: a technique or method for exploiting a vulnerability
- Attack: realization of a threat
- Mitigation: a technique for making an attack less likely, more expensive, or less valuable to an attacker.

Security isn't always free

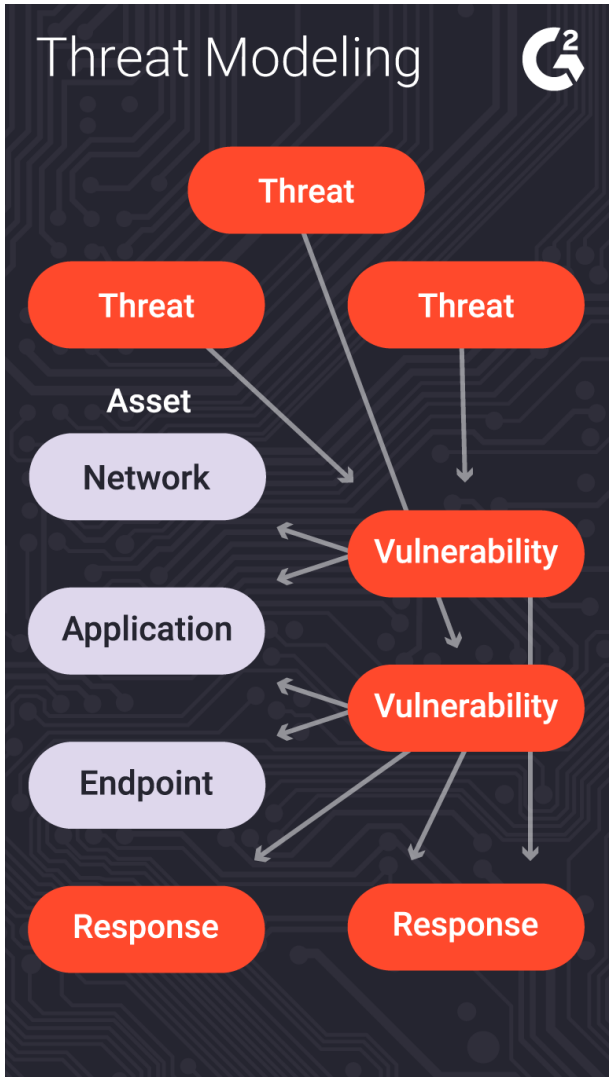
- In software, as in the real world…
- You just moved to a new house, someone just moved out of it. What do you do to protect you belongings/property?
- Do you change the locks?
- Do you buy security cameras?
- Do you hire a security guard?
- Do you even bother locking the door?

Security is about managing risk

- Increasing security might:
  - Increase development & maintenance cost
  - Increase infrastructure requirements
  - Degrade performance
- But, if we are attacked, increasing security might also:
  - Decrease financial and intangible losses
- How likely do we think we are to be attacked in some particular way?

Threat modeling can help us analyze the issues

- What is being defended?
- What malicious actors exist and what attacks might they employ?
- What value can an attacker extract from a vulnerability?
- Who do we trust? What parts of the system do we trust?
- What can we do in case of attack?



A Baseline Threat Model

- Trust:
  - Developers writing our code (at least for the code they touch)
  - Server running our code
  - Popular dependencies that we use and update
- Don’t trust:
  - Code running in browser
  - Inputs from users
  - Other employees (employees should have access only to the resources they need)

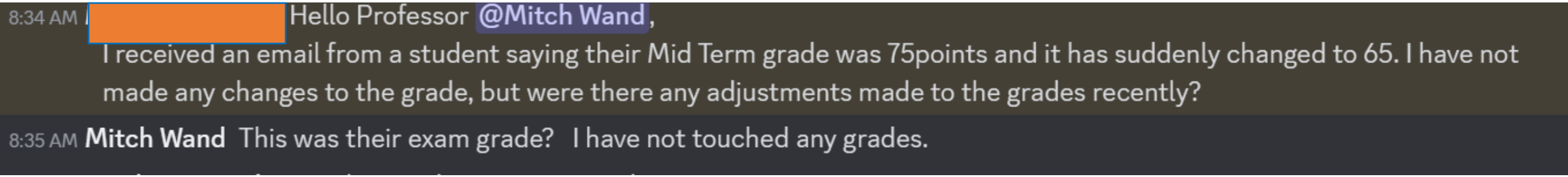
Man vs. Musk: A Whistleblower Creates Headaches for Tesla

An employee who was fired after expressing safety concerns leaked personnel records and sensitive data about driver-assistance software.

A Baseline Security Policy

- Encrypt all data in transit, sensitive data at rest
- Use multi-factor authentication
- Use encapsulated zones/layers of security
  - Different people have access to different resources
  - Principle of Least Privilege
- Log everything! (employee data accesses/modifications) (maybe)
- Do regular, automatic, off-site backups
- Bring in security experts early for riskier situations

How much should you log?



Backups can mitigate the risks of a ransomware attack

Off-site backups mitigate the risks of natural disasters

In the remainder of this module, we will discuss 5 major classes of vulnerabilities

- Vulnerability 1: Code that runs in an untrusted environment
- Vulnerability 2: Untrusted Inputs
- Vulnerability 3: Bad authentication (of both sender and receiver!)
- Vulnerability 4: Malicious software from the software supply chain
- Vulnerability 5: Failure to apply security policy.

https://owasp.org/www-project-top-ten/

Vulnerability 1 Example: authentication code in a web application

Curses! Foiled Again!

Front End

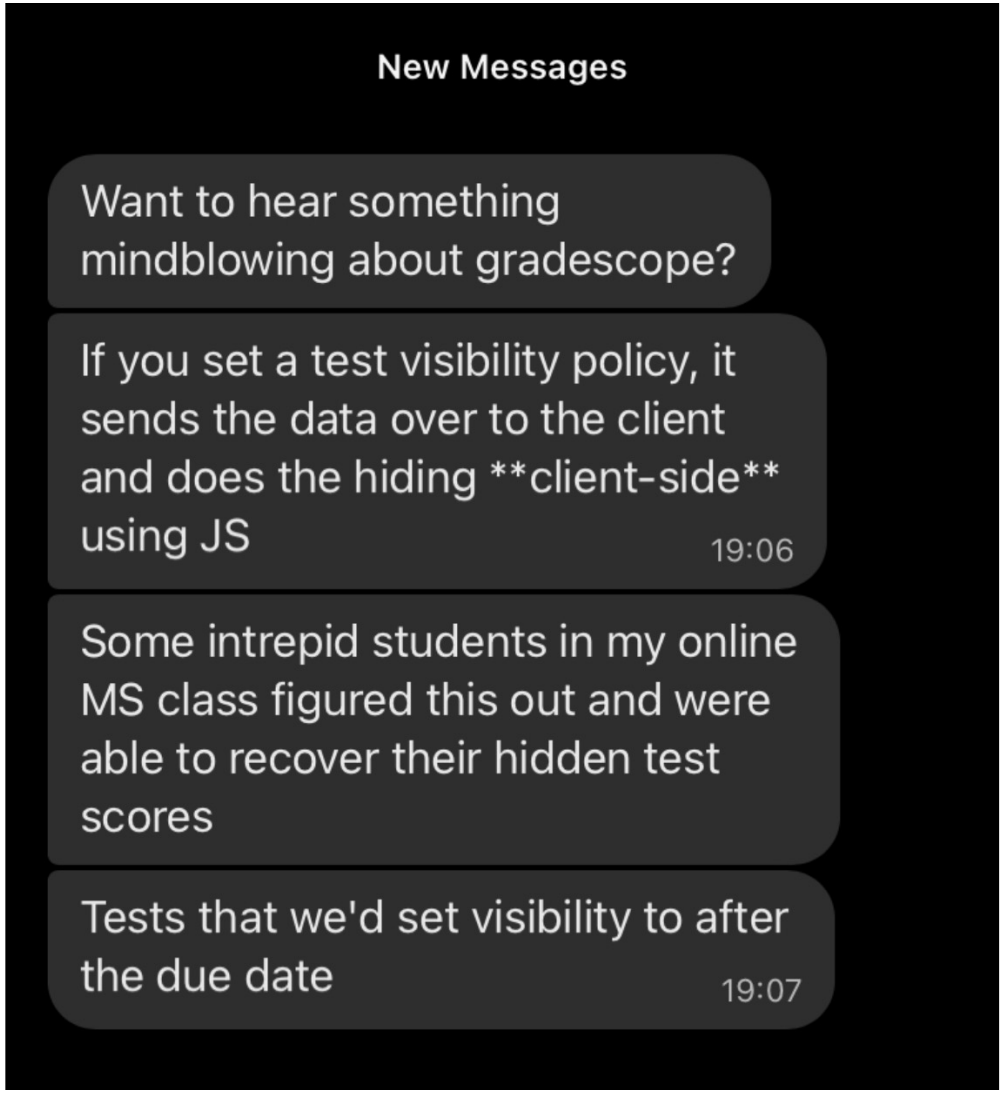
```
function checkPassword(inputPassword:
string){
  if(inputPassword === 'letmein'){
    return true;
  }
  return false;
}
```

Trust boundary

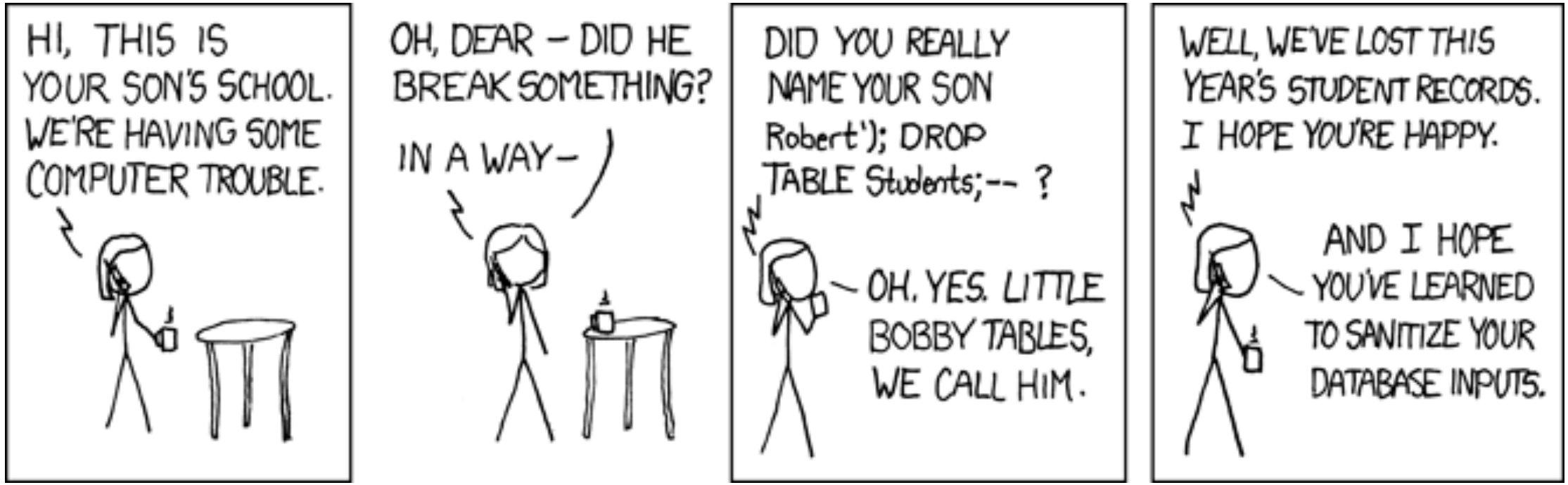
Fix: Move code to back end (duh!)

Back End

Who would do such a silly thing?



Vulnerability 2: Data controlled by a user flowing into our trusted codebase



https://xkcd.com/327/

Example: code injection

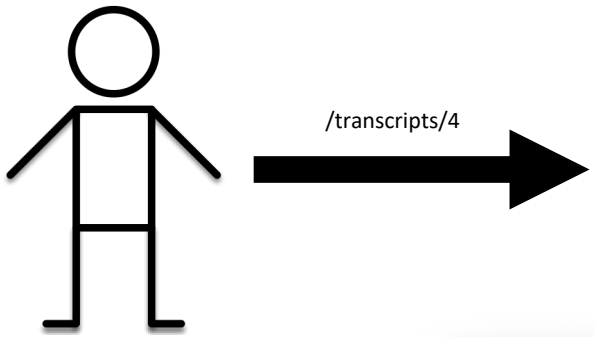
```
String query = "SELECT * FROM accounts WHERE
       name='" + request.getParameter("name") + "'";
```

Parameter name	Constructed Query	Effect
Alice	SELECT * FROM accounts WHERE name='Alice';	Select a single account
Alice O'Neal	SELECT * FROM accounts WHERE name='Alice O'Neal';	SQL Error
5' OR '1'='1	SELECT * FROM accounts WHERE name='5' OR '1'='1';	**Select all accounts**

OOPS!

- OWASP A03:2021-Injection

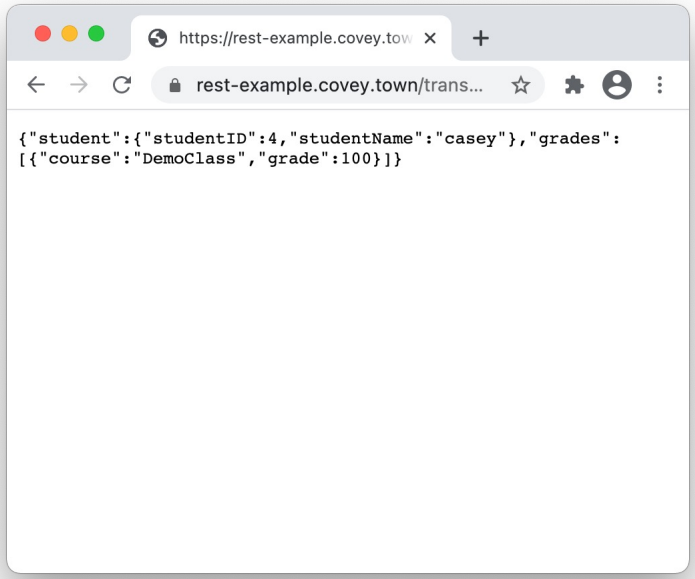
Example: Cross-site scripting (XSS)



/transcripts/4

Trusted Server

```
app.get('/transcripts/:id', (req, res) => {
  // req.params to get components of the path
  const {id} = req.params;
  const theTranscript = db.getTranscript(parseInt(id));
  if (theTranscript === undefined) {
    res.status(404).send(`No student with id = ${id}`);
  }
  {
    res.status(200).send(theTranscript);
  }
});
```



Example: Cross-site scripting (2)

/transcripts/%3Ch1%3e...

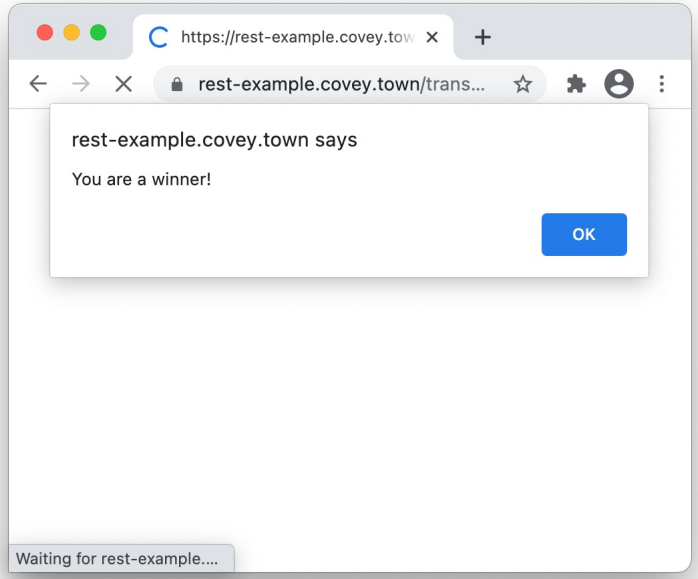
Trusted Server

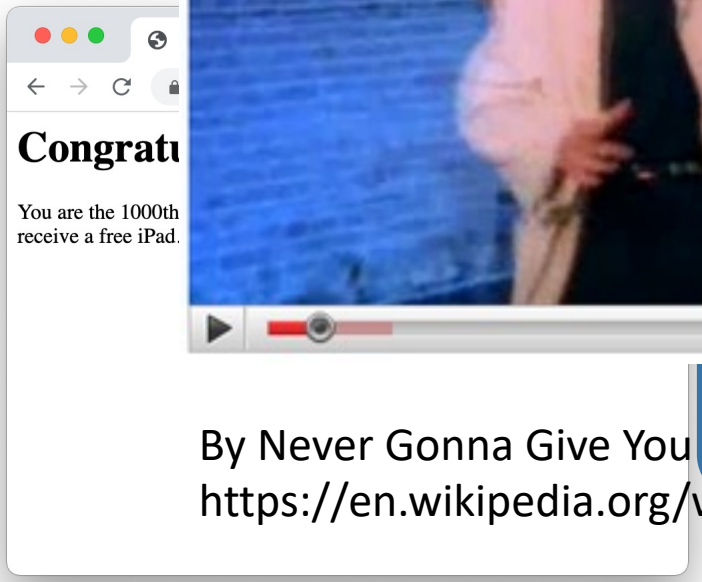
```
app.get('/transcripts/:id', (req, res) => {
  // req.params to get components of the path
  const {id} = req.params;
  const theTranscript = db.getTranscript(parseInt(id));
  if (theTranscript === undefined) {
    res.status(404).send(`No student with id = ${id}`);
  }
  {
    res.status(200).send(theTranscript);
  }
});
```

```
<h1>Congratulations!</h1>
  You are the 1000th visitor to the transcript site! You have been selected
to receive a free iPad. To claim your prize <a
href='https://www.youtube.com/watch?v=DLzxrzFCyOs'>click here!</a>
  <script language="javascript">
document.getRootNode().body.innerHTML=
'<h1>Congratulations!</h1>You are the 1000th visitor to the transcript site!
You have been selected to receive a free iPad. To claim your prize <a
href="https://www.youtube.com/watch?v=DLzxrzFCyOs">click here!</a>';
alert('You are a winner!');
</script>
```

Example: Cross-site scripting (3)

/transcripts/%3Ch1%3e...






```
app.get('/transcripts/:id', (req, res) => {
                                              the path
                                              ipt(parseInt(id));
                                              with id = ${id}` );
                                              or to the transcript site! You
                                              ive a free iPad. To claim your
                                              com/watch?v=DLzxrzFCyOs'>click
                                              ipt">
                                              /.innerHTML=
                                              You are the 1000th visitor to
                                              ave been selected to receive a
                                              ize <a
                                              com/watch?v=DLzxrzFCyOs">click
alert('You are a winner!');
</script>
```

By Never Gonna Give You Up music video., Fair use, https://en.wikipedia.org/w/index.php?curid=22192466

A code injection attack (in Apache struts) cost Equifax $1.4 Billion

NEWS

Equifax Says Cybersecurity Breach Has Cost $1.4 Billion

EMMA HURT • MAY 10, 2019

CVE-2017-5638 Detail

Current Description

The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 has incorrect exception handling and error-message generation during file-upload attempts, which allows remote attackers to execute arbitrary commands via a crafted Content-Type, Content-Disposition, or Content-Length HTTP header, as exploited in the wild in March 2017 with a Content-Type header containing a #cmd= string.

The Log4J code injection vulnerability compromised many networks in 2021

Extremely Critical Log4J Vulnerability Leaves Much of the Internet at Risk

December 10, 2021 Ravie Lakshmanan

The Apache Software Fou
actively exploited zero-da
Apache Log4j Java-based
execute malicious code a
systems.

CVE-2021-44228 Detail

Current Description

Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.
https://nvd.nist.gov/vuln/detail/CVE-2021-44228

Mar 8, 2022

APT41 COMPROMISED SIX U.S. STATE

The APT41 group compromised at least six U.S. state government networks between May and February in a "deliberate campaign" that reflects new attack vectors and retooling by the prolific Chinese state-sponsored group.

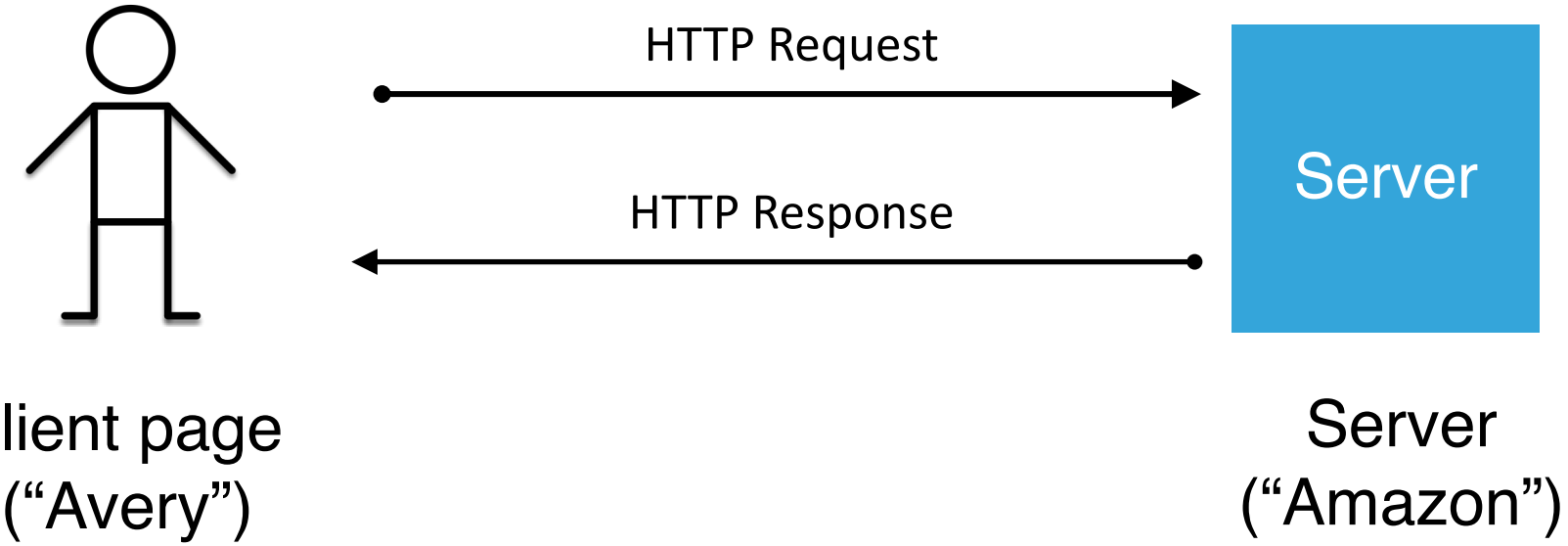
https://thehackernews.com/2021/12/extremely-critical-log4j-vulnerability.html

https://duo.com/decipher/apt41-compromised-six-state-government-networks

Mitigating against code injection attacks

- Use tools like TSOA to automatically generate safe code.
- Manually sanitize inputs to prevent them from being executable
- Avoid unsafe query languages (e.g. SQL, LDAP, language-specific languages like OGNL in java). Use “safe” subsets instead.
- Avoid use of languages (like C or C++) that allow code to construct arbitrary pointers or write beyond a valid array index
- eval() in JS – executes a string as JS code

Vulnerability 3: Bad Authentication



- How does Amazon know that this request is coming from Avery?
- How does Alice know that this request is coming from Amazon?

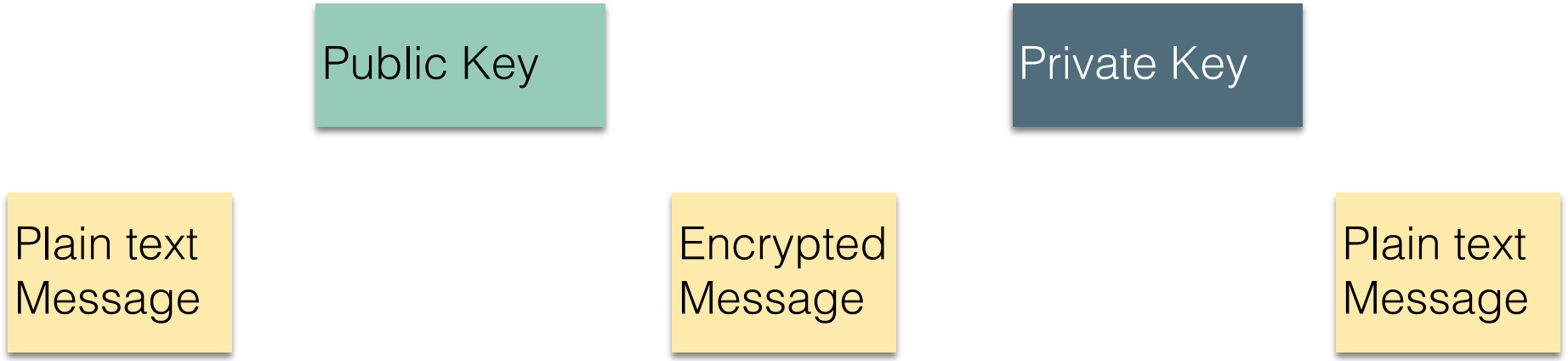
How does Amazon know that this request is coming from Avery?

- Password
  - Establishes that the request is coming from someone who knows Avery's password
- 2-factor authentication
  - Something the user has (physical key, bank card)
  - Something the user knows (password, PIN)
  - Something the user is (biometrics, address history, etc.)

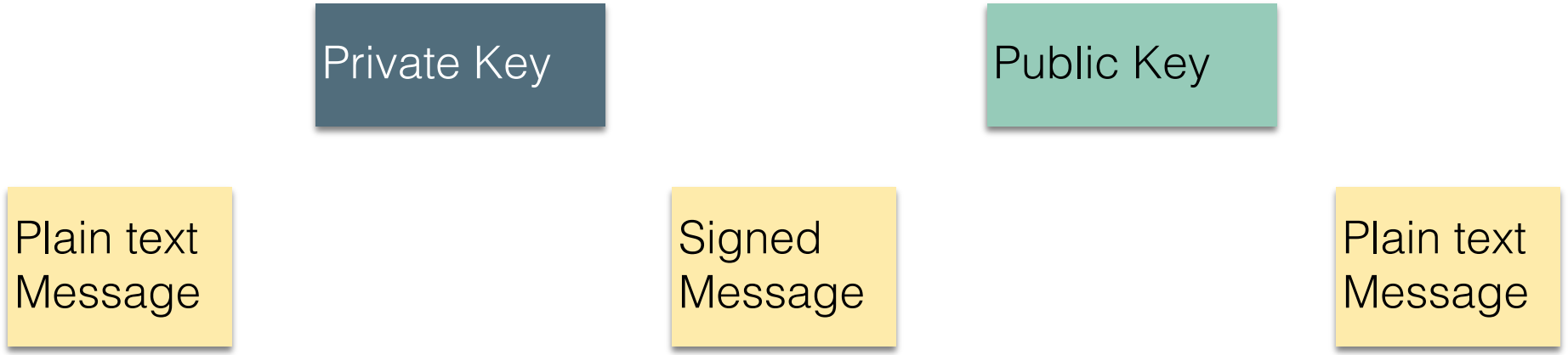
How does Avery know that this request is coming from Amazon?

- SSL is a protocol for encryption that uses *asymmetric* cryptography
- Each party has a *public* key and a *private* key
- Messages *encrypted* with a given public key can only be decrypted by matching private key
- Messages *signed* with a given private key can be validated by anyone with the public key
- A third-party can *endorse* that a public key is held by an entity and produce a *certificate*

Encrypt messages with a public key to ensure confidentiality



Encrypt messages with a private key to ensure integrity



Certificate Authorities associate public keys with real-world entities

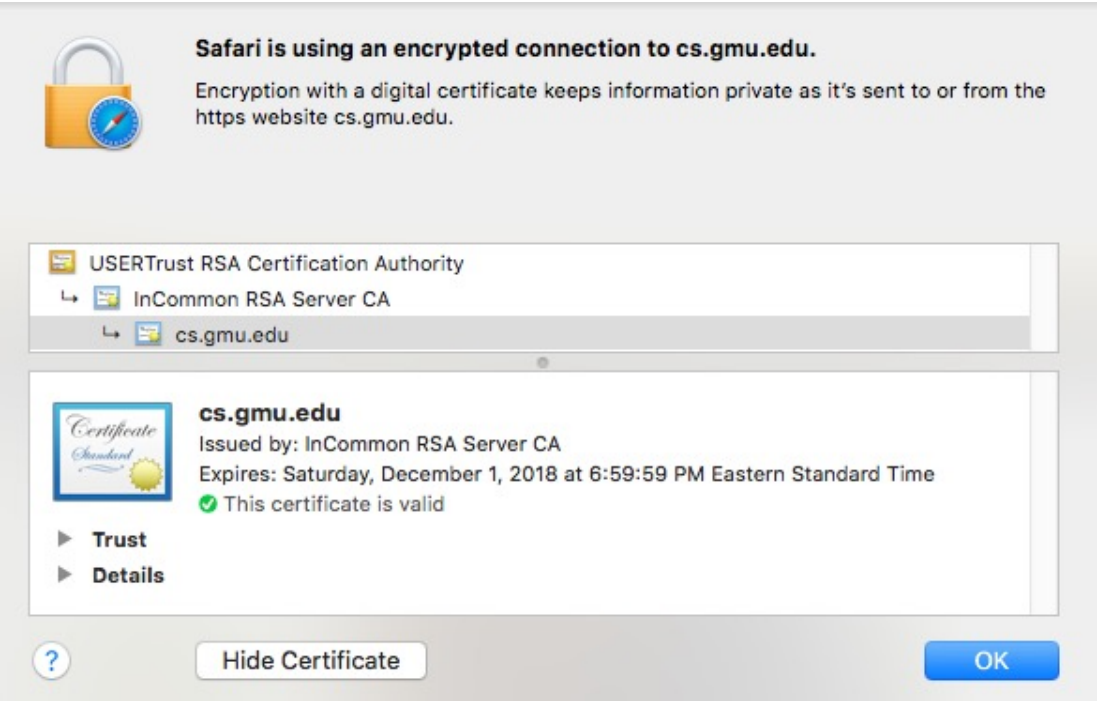
- CA's are trusted entities (their public keys are distributed along with your OS).
- To acquire a certificate, Amazon.com will share their public key and some real-world proof that they are amazon.com to the CA.
- The CA locks Amazon's public key with its own private key. This is called a "certificate".
- When we visit amazon.com, it presents its certificate to our browser.
- Our browser unlocks the certificate with the CA's public key, thus getting amazon's public key.
- Because we trust the CA, we can trust that this public key is really Amazon.com .

Integrity!

Certificate Authorities issue SSL Certificates

Certificate Authorities are Implicitly Trusted

- For this to work, we had to already know the CA's public key
- There are a small set of “root” CA’s (think: root DNS servers)
- Every computer/browser is shipped with these root CA public keys



What happens if a CA is compromised, and issues invalid certificates?

Security

Comodo-gate hacker brags about forged certificate exploit

Tiger-blooded Persian cracker boasts of mighty exploits

Security

Fuming Google tears Symantec a new one over rogue SSL certs

We've got just the thing for you, Symantec ...

By Iain Thomson in San Francisco 29 Oct 2015 at 21:32 36 SHARE ▼

Google has read the riot act to Symantec, scolding the security biz for its

You can do this for your website for free

- letsencrypt.com

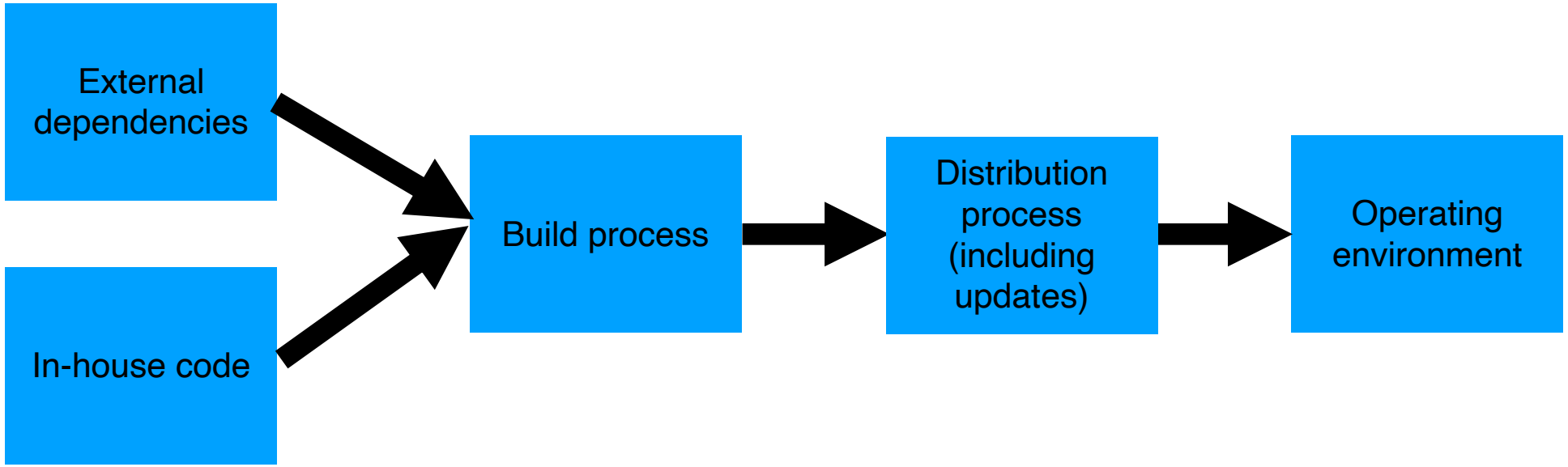
Other mitigations for access-control threats

- Implement multi-factor authentication
- Make sure passwords are not weak, have not been compromised.
- Apply per-record access control
  - Principle of least privilege
- Harden pathways for account creation, password reset.
- Use an expert vendor, like Auth0, to handle login
  - They might do it better than you can.

Vulnerability 4: Supply-Chain Attacks

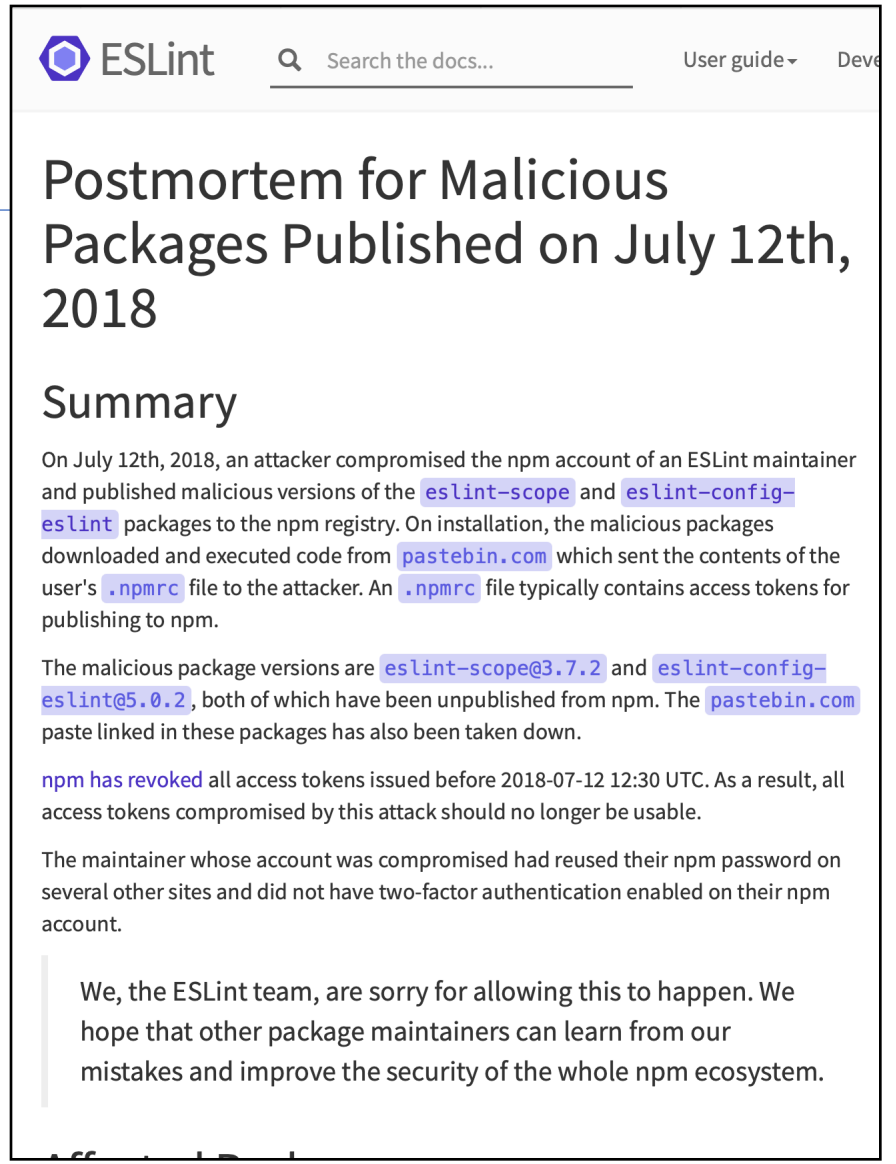
- Do we trust our own code?
- Third-party code provides an attack vector

The software supply chain has many points of weakness



Example: the eslint-scope attack (2018)

- On 7/12/2018, a malicious version of eslint-scope was published to npm.
- eslint-scope is a core element of eslint, so many many users were affected.
- Let’s analyze this…



ESLint Search the docs... User guide Deve

Postmortem for Malicious Packages Published on July 12th, 2018

Summary

On July 12th, 2018, an attacker compromised the npm account of an ESLint maintainer and published malicious versions of the `eslint-scope` and `eslint-config-eslint` packages to the npm registry. On installation, the malicious packages downloaded and executed code from `pastebin.com` which sent the contents of the user's `.npmrc` file to the attacker. An `.npmrc` file typically contains access tokens for publishing to npm.

The malicious package versions are `eslint-scope@3.7.2` and `eslint-config-eslint@5.0.2`, both of which have been unpublished from npm. The `pastebin.com` paste linked in these packages has also been taken down.

npm has revoked all access tokens issued before 2018-07-12 12:30 UTC. As a result, all access tokens compromised by this attack should no longer be usable.

The maintainer whose account was compromised had reused their npm password on several other sites and did not have two-factor authentication enabled on their npm account.

> We, the ESLint team, are sorry for allowing this to happen. We hope that other package maintainers can learn from our mistakes and improve the security of the whole npm ecosystem.

https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes/

This incident leveraged several small security failures

- An eslint-scope developer used their same password on another site.
- The other site did not use 2FA
- Password was leaked from the other site.
- Attacker created malicious version of eslint-scope
- Many users did not use package-lock.json, so their packages automatically installed the new (evil) version.
- The malicious version sent copies of the user's .npmrc to the attacker. This file typically contains user tokens.
- Estimated 4500 tokens were leaked and needed to be revoked.

Example: the SolarWinds attack (2020)

- Many networks compromised
- Not discovered for months

PODCASTS

HARD LESSONS OF THE SOLARWINDS HACK

Cybersecurity reporter Joseph Menn on the massive breach the US didn't see coming

By Nilay Patel | @reckless | Jan 26, 2021, 9:13am EST

SHARE

In December, details came out on one of the most massive breaches of US cybersecurity in recent history. A group of hackers, likely from the Russian government, had gotten into a network management company called SolarWinds and infiltrated its customers' networks. This access was then used to breach everything from Microsoft to US government agencies, including the US Treasury and departments of Homeland Security, State, Defense, and Commerce.

This problem was recognized ages ago

- Ken Thompson (the Unix guy) - **1984**
- Showed how to plant a bug in a compiler, so that any program compiled by that compiler would contain a backdoor.

The final step is represented in Figure 3.3. This simply adds a second Trojan horse to the one that already exists. The second pattern is aimed at the C compiler. The replacement code is a Stage I self-reproducing program that inserts both Trojan horses into the compiler. This requires a learning phase as in the Stage II example. First we compile the modified source with the normal C compiler to produce a bugged binary. We install this binary as the official C. We can now remove the bugs from the source of the compiler and the new binary will reinsert the bugs whenever it is compiled. Of course, the login command will remain bugged with no trace in source anywhere

TURING AWARD LECTURE

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

INTRODUCTION

I thank the ACM for this award. I can't help but feel that I am receiving this honor for timing and serendipity as much as technical merit. UNIX[1] swept into popularity with an industry-wide change from central mainframes to autonomous minis. I suspect that Daniel Bobrow [1] would be here instead of me if he could not afford a PDP-10 and had had to "settle" for a PDP-11. Moreover, the current state of UNIX is the result of the labors of a large number of people.

There is an old adage, "Dance with the one that brought you," which means that I should talk about UNIX. I have not worked on mainstream UNIX in many years, yet I continue to get undeserved credit for the work of others. Therefore, I am not going to talk about UNIX, but I want to thank everyone who has contributed.

That brings me to Dennis Ritchie. Our collaboration has been a thing of beauty. In the ten years that we have worked together, I can recall only one case of miscoordination of work. On that occasion, I discovered that we both had written the same 20-line assembly language program. I compared the sources and was astounded to find that they matched character-for-character. The result of our work together has been far greater than the work that we each contributed.

I am a programmer. On my 1040 form, that is what I put down as my occupation. As a programmer, I write programs. I would like to present to you the cutest program I ever wrote. I will do this in three stages and try to bring it together at the end.

STAGE I

In college, before video games, we would amuse ourselves by posing programming exercises. One of the favorites was to write the shortest self-reproducing program. Since this is an exercise divorced from reality, the usual vehicle was FORTRAN. Actually, FORTRAN was the language of choice for the same reason that three-legged races are popular.

More precisely stated, the problem is to write a source program that, when compiled and executed, will produce as output an exact copy of its source. If you have never done this, I urge you to try it on your own. The discovery of how to do it is a revelation that far surpasses any benefit obtained by being told how to do it. The part about "shortest" was just an incentive to demonstrate skill and determine a winner.

Figure 1 shows a self-reproducing program in the C[2] programming language. (The purist will note that the program is not precisely a self-reproducing program, but will produce a self-reproducing program.) This entry is much too large to win a prize, but it demonstrates the technique and has two important properties that I need to complete my story: 1) This program can be easily written by another program. 2) This program can contain an arbitrary amount of excess baggage that will be reproduced along with the main algorithm. In the example, even the comment is reproduced.

[1] UNIX is a trademark of AT&T Bell Laboratories.

© 1984 0001-0782/84/0800-0761 75¢

A 2021 NCSU/Microsoft found that many of the top 1% of npm packages had vulnerabilities

- Package inactive or deprecated, yet still in use
- No active maintainers
- At least one maintainer with an inactive (purchasable) email domain
- Too many maintainers or contributors to make effective maintenance or code control
- Maintainers are maintaining too many packages
- Many statistics/combinations: see the paper for details.

"What are Weak Links in the npm Supply Chain?" By: Nusrat Zahan, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila, Laurie Williams https://arxiv.org/abs/2112.10165

Threat Mitigation: Process-based problems need process-based solutions

- External dependencies
  - Audit all dependencies and their updates before applying them
- In-house code
  - Require developers to sign code before committing, require 2FA for signing keys, rotate signing keys regularly
- Build process
  - Audit build software, use trusted compilers and build chains
- Distribution process
  - Sign all packages, protect signing keys
- Operating environment
  - Isolate applications in containers or VMs

Supply-chain risks include more than just software.

intertek
Total Quality. Assured.

Industries

Home / Industries & Services / Auditing / Business Assurance / Supply Chain Security

Supply Chain Security

In today's global marketplace, it is more important than ever to have a transparent view into your supply chain, no matter how remote suppliers may be from where you actually conduct your business. As a result, suppliers and manufacturers need solutions in place to demonstrate compliance in a number of areas dictated by today's business climate.

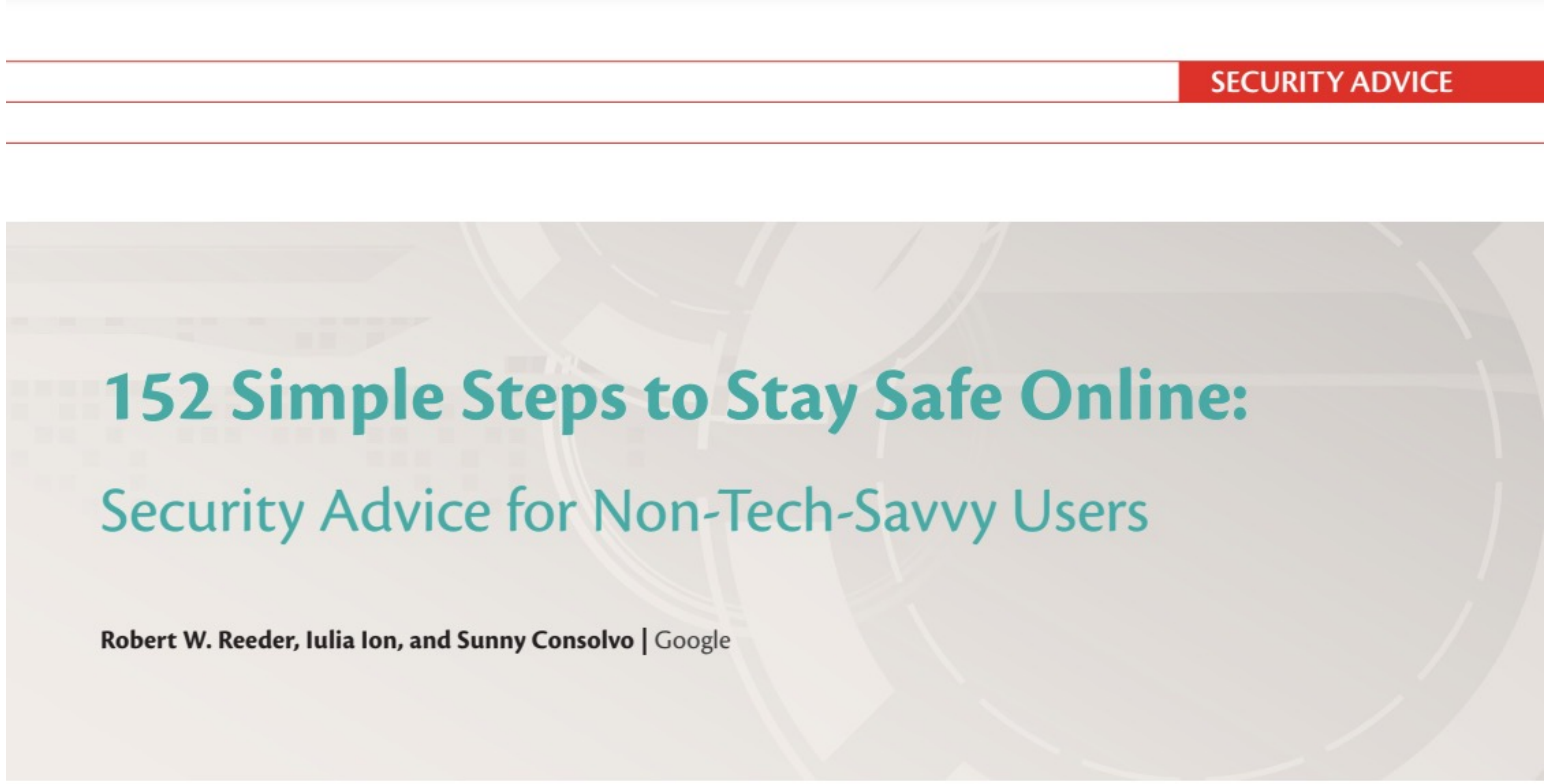
In order to demonstrate enforcement of and compliance to international supply chain security

Supply Chain Assessments - Using a series of risk-based assessment tools and audit solutions to evaluate and benchmark suppliers, supply chain assessments help global companies manage and track the performance in their supply chains. The assessments measure business risk, capacity and capabilities, workplace conditions, product quality and safety, security and environmental sustainability.

Your suppliers' risks are your risks.

- MOVEit is a file transfer program owned by Progress Software.
- Over 2500 organizations used the program to move sensitive personal data.
- They were attacked in May 2023.
- Prof. Wand says: my bank didn't use MOVEit, but they used a supplier who did.
- Now, they have to take expensive steps to offer me identity-protection services, etc.

Vulnerability 5: Failure to Apply Security Policy



IEEE Security & Privacy 15:5 (2017)

Other mitigations for access-control threats

- Implement multi-factor authentication
- Make sure passwords are not weak, have not been compromised.
- Apply per-record access control
  - Principle of least privilege
- Harden account creation, password reset pathways
- Use an expert vendor, like Auth0, to handle login
  - They can do it better than you can.

But how do you get your developers to do all this?

Outline of this lecture

1. Definition of key vocabulary
2. Some common vulnerabilities, and possible mitigations
3. Getting security right is about people as well as software.

David Blank-Edelman (former head of Systems at Khoury)

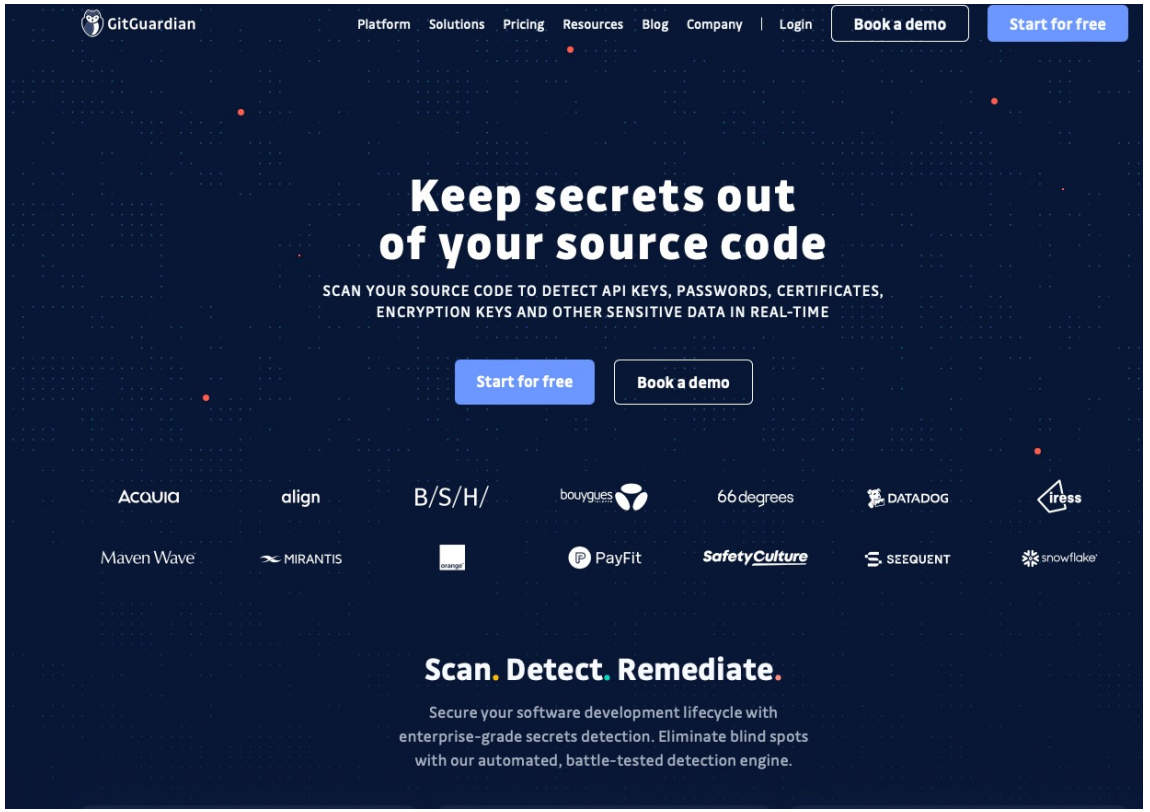
"The solution is in front of the screen, not behind it"

A security architecture must include a security culture

- Security architecture is a set of mechanisms and policies that we build into our system to mitigate risks from threats
- Vulnerability: a characteristic or flaw in system design or implementation, or in the security procedures, that, if exploited, could result in a security compromise
- Threat: potential event that could compromise a security requirement
- Attack: realization of a threat

Example mechanism: secret detection

- Recall: SSL only is effective if the private key… remains private
- Applications may have many other *secret* values (e.g. access tokens for other services)
- Tools like *GitGuardian* automatically detect secrets in repositories



Mechanisms aren't enough: Do developers keep secret keys secret?

- Industrial study of secret detection tool in a large software services company with over 1,000 developers, operating for over 10 years
- What do developers do when they get warnings of secrets in repository?
  - 49% remove the secrets; 51% bypass the warning
- Why do developers bypass warnings?
  - 44% report false positives, 6% are already exposed secrets, remaining are "development-related" reasons, e.g. "not a production credential" or "no significant security value"

Is it a management problem or a tool problem?

"Why secret detection tools are not enough: It's not just about false positives - An industrial case study"
Md Rayhanur Rahman, Nasif Imtiaz, Margaret-Anne Storey & Laurie Williams https://link.springer.com/article/10.1007/s10664-021-10109-y

Elements of a security culture

- Make security a regular part of the process.
  - Include security tools as part of the build/release process
  - Tools may have false positives and false negatives
  - Educate developers about when how to recognize positives that look false, but aren’t
  - Include security review as regular part of code review

Learning Objectives for this Module

- You should now be able to:
  - Define key terms relating to software/system security
  - Describe some of the tradeoffs between security and other requirements in software engineering
  - Explain 5 common vulnerabilities in web applications and similar software systems, and describe some common mitigations for each of them.
  - Explain why software alone isn’t enough to assure security