ネットワークを介した様々な攻撃

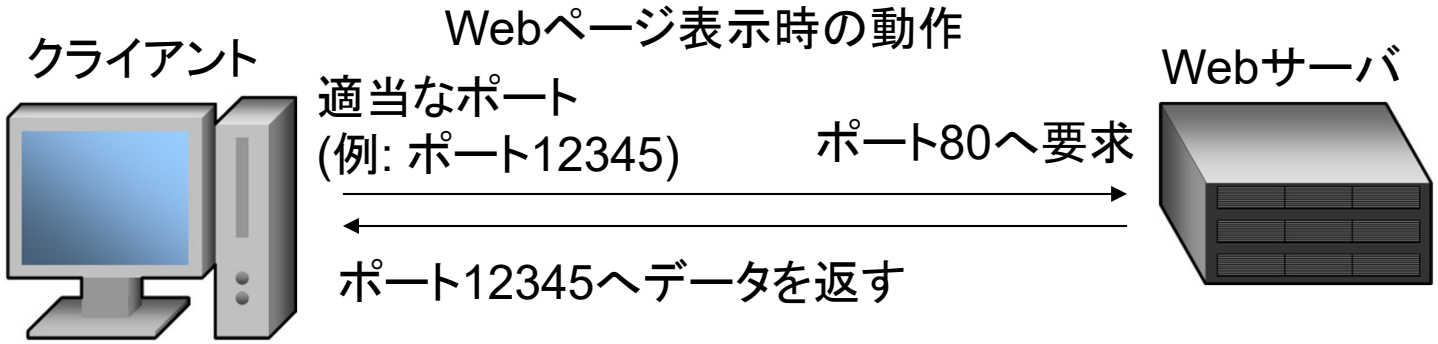
名古屋大学 情報基盤センター
情報基盤ネットワーク研究部門
基盤ネットワーク研究グループ
嶋田 創

概要

- クライアント-サーバ間の通信路での攻撃
- 公開サーバー般に対する攻撃
  - ポート公開中のサービスの脆弱性を狙った攻撃
  - サービス不能(Denial of Service)攻撃
- Webサービスの提供者/利用者への攻撃
  - Webサービス提供者側への攻撃
  - Webサービス利用者側への攻撃
- 認証機構と突破の試み

ポート番号とその公開の簡単な復習

- TCP/IPおよびUDP/IPの通信はお互いのポート(0-65535)へデータを送ることで成立
- 1023番までのポートはWell Known Portとして、特定のサービスに紐付けられている
- 一般的なTCP接続による通信
  1. 適当なポート番号を確保して、そこからサーバに接続要求を出す
  2. サーバ側は提示されたポートに、要求されたデータを返す
  - 最初の接続要求がうまくいかなければ、通信は成立しない
  - サーバ側はサービスに対応するポートで接続を待ち受ける



クライアントとサーバの間の通信にどこで攻撃する?

クライアント側を狙う

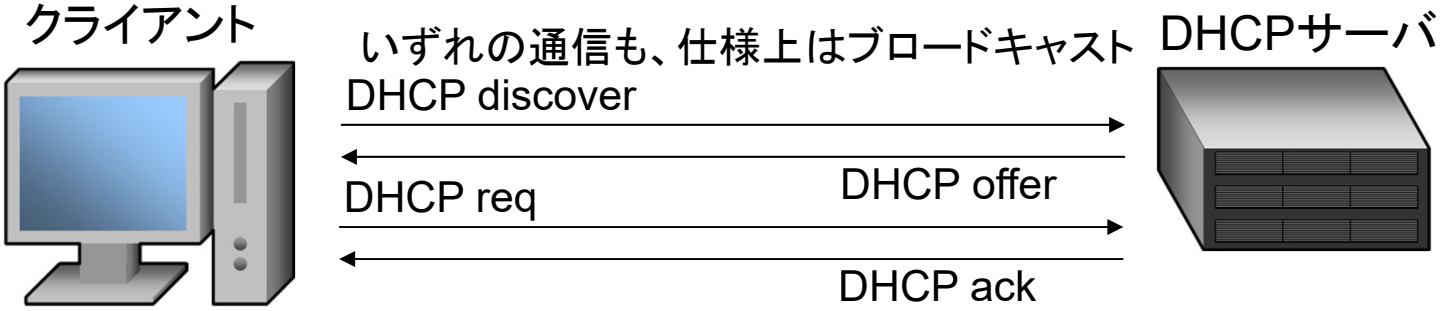
- (クライアントがネットワーク接続時に)
- クライアントがサーバに通信を開始する時に
- クライアントとサーバの間の経路上で
- Webサーバ上のコンテンツにクライアント攻撃用コンテンツをしかける

サーバ側を狙う

- ポート公開中のサービスの脆弱性を狙った攻撃
- サービス不能(Denial of Service)攻撃
- Webサーバ上のコンテンツにサーバ攻撃用のコンテンツを送り込む

DHCPでIPアドレスをわりふる時に(1/2)

- 最近はDHCPの利用が増えて自分でIPアドレスを設定することが減った
  - IPアドレスを固定する場合においても、DHCPサーバ側でMACアドレスと割り振るIPアドレスを固定
- DHCPの手続きはサブネット内へのブロードキャストが基本なので、偽DHCPサーバ設置は容易
- 対策
  - スイッチへのDHCP snoopingの設定
    - 正規DHCPサーバの接続ポートのみからDHCP応答を許可
  - サブネット内のブロードキャスト通信の監視



DHCPでIPアドレスをわりふる時に(2/2)

IPv6だと問題がさらに複雑に

- DHCPv6とRA(Router Advertisement)の2種類がある
  - OSによってどちらが優先されるか差がある(正しく実装されていない)
    - 古いバージョンでは片方のみ実装されているOSも
  - 偽のDHCPv6や偽RAによる誘導はIPv4よりやりやすいのではという印象
- DHCP spoofingと同様、スイッチ側でRA guardやDHCPv6 shieldを設定することはできる
- そもそもIPv6が2020年あたりまで仕様を改廃していたりしたのでOS側が迷った印象
  - 今の最新OSはまず大丈夫だが、少し古めのOSが混ざった環境は?
- 東工大の北口先生がこの問題に詳しい[1]

[1] https://www.nic.ad.jp/ja/materials/iw/2017/proceedings/s03/s3-kitaguchi.pdf

DNSで名前解決する時に(1/3)

- 悪意のあるDNSを作り、名前解決時に本来のIPアドレスではない応答を返すことはできる
  - 例: www.nagoya-u.ac.jpの名前解決結果をどこかのクラウドサーバ上のIPアドレスにするとか(当然、そこには悪意のあるコンテンツが)
- 一部のブロードバンドルータを狙うマルウェアでは、DNSの設定を書き換えるものがある
- そもそも、一部の(国外の)ISPやフリーWiFiではDNSの応答で特定サービスに誘導することも行われていたり…
- 対策: (ネットワークが信頼できない時には)信頼できるPublic DNSを手動で指定
  - 例: 8.8.8.8, 8.8.4.4のGoogle Public DNS
- そもそもDNSは暗号化されていないUDP通信なので、保護しにくい

DNSで名前解決する時に(2/3)

- 対策としてDNSSECは提案されているが普及が微妙
  - DNSサーバ側がDNSSECに対応しないといけないが、普及が…
- 代わりに、最近のWebブラウザでは、DNS over HTTPS(DoH)のサポートが進んでいる
  - HTTPSを使って信頼できるDNSサーバまで暗号化通信路で名前解決を投げる
  - Firefoxが2021/7にカナダで有効化したらしい
  - OS側でも対応する話が出ている
  - 欠点
    - 名前解決というプライバシーに直結する情報が、特定のDNSサーバに集約されてしまう
    - 集約先のDNSサーバ群が特定の名前解決を一斉に拒否したら?(Webにおけるgoogle村八分話のように)

DNSで名前解決する時に(3/3)

- DNSのキャッシュに偽名前解決を注入する、DNS cache poisoningという攻撃もある
  - 毎度毎度ネットワークのはるか遠くのDNSサーバまで名前解決に行くのは面倒 →たいていのDNSサーバはキャッシュ機能がある
    - DNS側もキャッシュすることを前提に、指定したTTL(Time To Live)の間はキャッシュして良いことになっている
    - サーバの引っ越し(IPアドレス)をする時には、事前にTTLを短くしておくおとを忘れずに
  - DNSサーバが遠隔のDNSサーバに問い合わせた時に、偽の応答を返して偽名前解決を注入
    - DNSの仕様で問い合わせ時のIDが16bitしか無いのでぶつけやすい
    - 注入するDNSサーバに名前解決の依頼を出しながら注入を試みるKaminsky攻撃が有名

偽無線LANアクセスポイントで(1/2)

- 全く同じSSIDの無線APを立てることは容易(Evil Twin攻撃)
  - 正規無線LANへの接続中の端末にdeauthenticationパケットやチャンネル変更通知(5GHz帯のDFS対応)を投げることも併用して
  - 全ての通信を特定サーバに誘導もできるし、DNS応答で特定のサーバのみ誘導とかもできる
    - まあ、「TLSが成立しない(サーバ証明書がおかしいとか)」とかで見つけることはできるが…
  - 個人的には、Free WiFi系は一旦安全なサーバにVPNなどを通してから使っている
- 特に、IDとパスワードを入力させる802.1x認証な無線LANインフラではID/パスワード窃取につながる

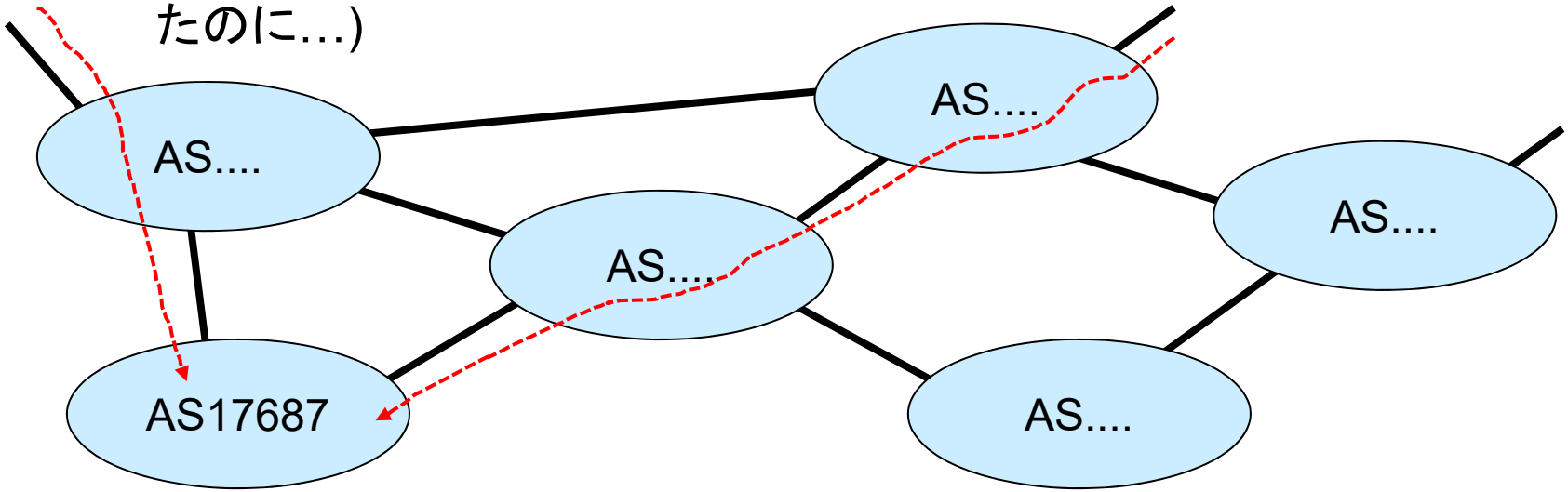
偽無線LANアクセスポイントで(2/2)

- 対策
  - 802.1x認証では正規の認証サーバの証明書情報を公開して、接続時に確認を促す
    - NUWNET/名大eduroamは公開中 (学内専用ページで)

      https://icts.nagoya-u.ac.jp/ja/services/nuwnet/
  - PSK認証では対策はほぼ無い?
    - フリーWiFiなどpre shared keyが公開されている所は、攻撃者も同じpre shared keyを偽無線アクセスポイントに設置するだろう
    - TLSへのman in the middleをなどをされた時に、「TLSが怪しい」通知が出たら気づくぐらいしかできない?
      - Zoomを常時立ち上げていると、フリーWi-Fiのcaptive portalで通信が捻じ曲げられると「安全な通信を確立できません」と出るのでカナリアに使える?

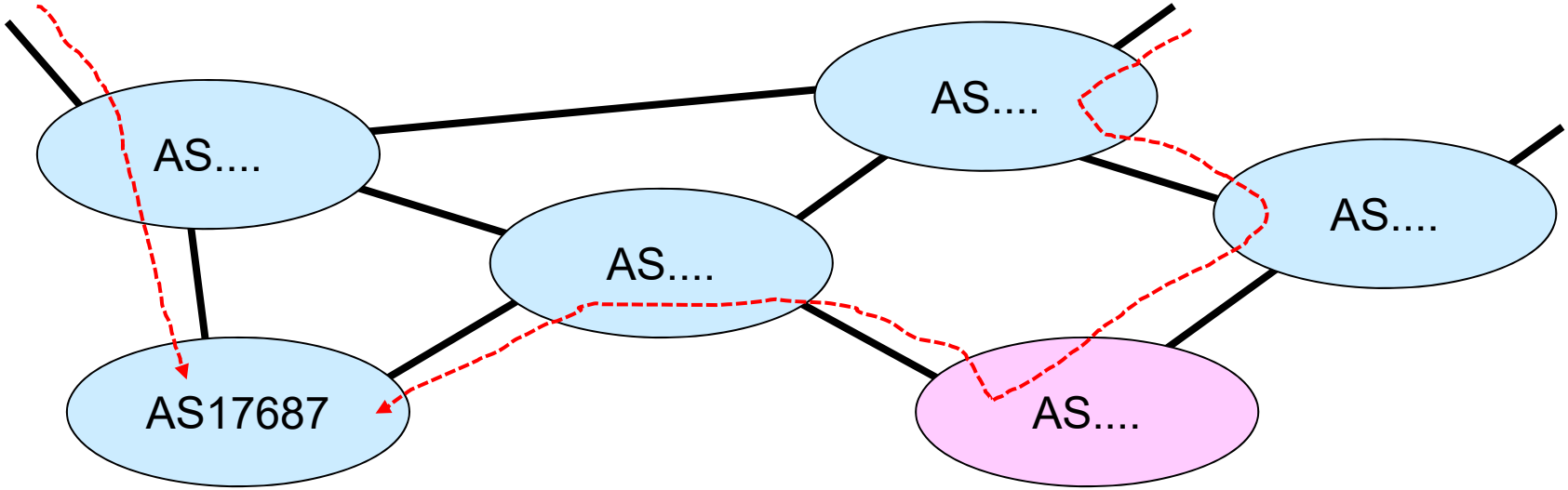
ネットワーク経路ハイジャック(BGP経路ハイジャック)

- BGP(Border Gateway Protocol)の動作
  - サブネットを運営する組織はAutonomous System(AS)番号を持つ
    - 名古屋大学のAS番号は17687
  - ASの番号を列挙したものが経路となる
    - ブロードキャストしたAS番号に経由したASのAS番号をつなげていく
    - 「どこに」「どこまで」ブロードキャストするかの制御も可能
  - 現在はIPv4で80万-90万経路ぐらい(10年ぐらい前は50万ぐらいだったのに…)



ネットワーク経路ハイジャック(BGP経路ハイジャック)

- このBGPに偽の経路を流す
  - 自分が存在するASの別サーバに誘導
  - 自分が存在するASの自分が乗っ取ったネットワーク機器を経由するようにルート変更
    - 暗号化されていない通信を見ることができる
    - 暗号化されている通信でも、「どことどこが、どれだけの頻度で、どれぐらいの通信量で通信している」という情報は得ることができる

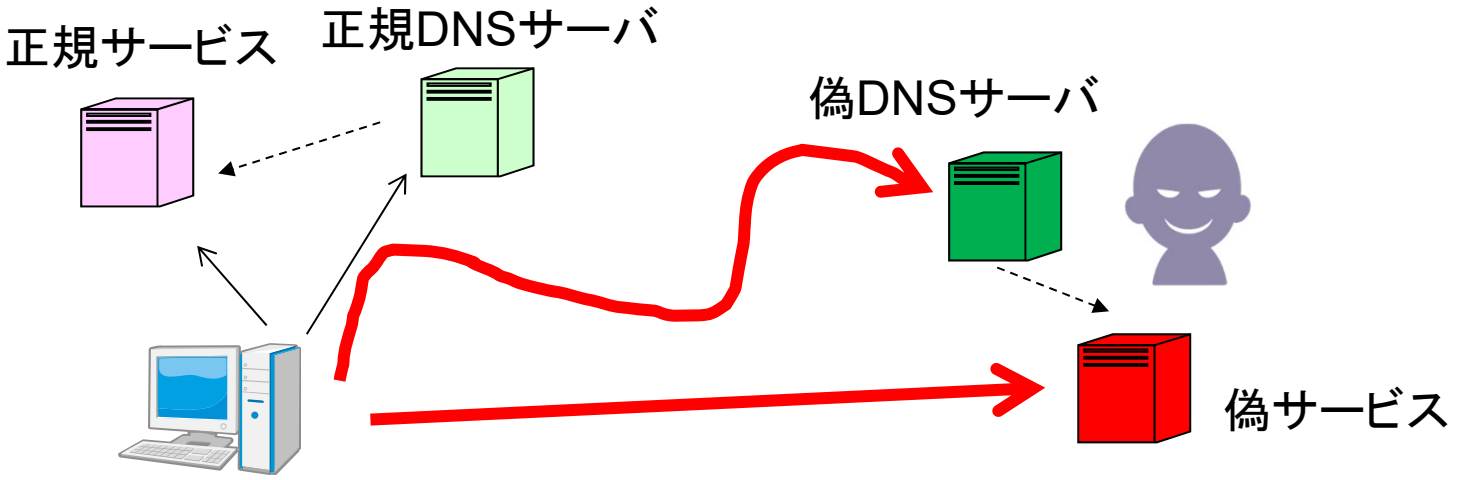


ハイジャック系の実例

DNSへのBGPハイジャックと偽DNSサーバによる仮想通貨窃取(2018/4)[1]

- BGPハイジャックでAmazonのDNSサービスであるRoute 53への経路をハイジャック

  →Route 53利用者が偽DNSサーバに誘導される
- 偽DNSが仮想通貨ウォレットサービスを偽サイトに誘導



[1] https://japan.zdnet.com/article/35118344/

ドメインハイジャック系(1/2)

- DNSインジェクション、ドメイン登録情報不正書き換え、などで実施
- ドメイン登録情報不正書き換え
  - ドメインレジストラを経由した登録において、そのドメインを管轄するDNSサーバを変更して自サーバへ誘導

(WHOIS情報)	[住所] 北区大深町4番20号 グランフロント大阪 タワーA35階 [Postal Address] osaka osaka 35F,4-20,ofukacho,kitaku [電話番号] +81.663764800 [FAX番号]	変　更
ネームサーバ1	NS0.MYDNS.JP	
ネームサーバ2	NS1.MYDNS.JP	
ネームサーバ3	NS2.MYDNS.JP	変　更
ネームサーバ4		

  - 多くのドメインレジストラでは、Webインタフェースで設定可能

    →WebインタフェースへのログインID/パスワード窃取して書き換え

ドメインハイジャック系(2/2)

- 有名サイトと1文字違いなど、typo違いを狙うのもドメインハイジャックの亜種と言えなくもない
  - domain squattingにちなんでtypo squattingと呼ぶ
    - …と思っていたら、ドッペルゲンガードメインなんて名前も使われ出した
  - 対策のために、名前の短いドメイン名(手打ちされやすい)はtypoするドメインも取得されていたりする
  - フィッシングにおいても多用される
  - 国際化ドメイン名(IDN: Internationalized Domain Name)になってくるともっと複雑になる…が、まだまだ普及していないのが救い?
    - 国際化ドメイン名では日本語も含め、Unicodeの文字を利用可能
    - 複雑なユニコード文字列は正規化処理が入ったりする
      - 例: ｶﾛﾘｰ → カロリー
    - Punycodeで等価なASCII文字のみのドメイン名に変換されたりする
      - 例: 日本語.jp <-> xn--wgv71a119e.jp

自動リンクの実装は安全ですか?

- 最近は、URLと解釈される文字列を自動的にハイパーリンクにする(Web)アプリが多い
  - こいつの実装はいい加減なものもそこそこ見られる(下のように、変にリンクされる事例もよく見られる) →セキュリティリスクは無いか?

詳細はhttp://example.comをご覧ください。また、～

- 国際化ドメイン名(IDN: Internationalized Domain Name)にUnicode文字列の正規化処理が入った時に安全か?
  - http://evil.c℀.example.comがhttp://evil.ca/c.example.comと解釈されるHostSplit脆弱性
- URLのクエリ部とかが別URLとしてリンクが生成されないか?

詳細はhttp://example.com/hoge?q=e`evil.com。

アップデートハイジャック系

- ソフトウェアアップデートの通信先にマルウェアを置く
  - アップデートサーバ乗っ取り、DNS乗っ取り、などで
- 2019/4のASUS Live Updateのハイジャックなど
- 例: EmEditorアップデートハイジャックを利用した攻撃[1]
  - 以下の様な.htaccessファイルがアップデート配布ディレクトリに置いてあった
  - 指定したIPアドレスの範囲からアップデート要求があれば別ファイルを配布

```
SetEnvIf Remote_Addr “106¥.188¥.131¥.[0-9]+” install
SetEnvIf Remote_Addr “133¥.6¥.94¥.[0-9]+” install
(… 同様に70行 …)
SetEnvIf Remote_Addr “124¥.248¥.207¥.[0-9]+” install
RewriteEngine on
RewriteCond %{ENV:install} =1
RewriteRule (.*¥.txt)$ /pub/rabe/editor.txt [L]
```

[1] https://jp.emeditor.com/general/今回のハッカーによる攻撃の詳細について/

Webブラウザ側で悪意のある動作を起こすもの

- クロスサイトスクリプティング(XSS)
- クロスサイトリクエスト偽造(CSRF: Cross Site Request Forgery)
- Cookie等を使った利用者追跡
- セッションハイジャック
  - 認証済み情報(主にCookie)窃取

→「Webサービスの提供者/利用者への攻撃」の中で説明

接続時の暗号スイートは安全ですか?(1/2)

- 暗号に寿命があることは情報セキュリティとリテラシで話した
- 公開中のポートに接続する時の暗号スイートは安全ですか?(寿命が来た暗号を使ってないか?)
- 後述するShodanなどでは有効な暗号スイート一覧が出るので、管理の甘そうなサーバを明確化できそう
- 例1: WebサーバのHTTPSのTLS(SSL)は?
  - すでに大多数のブラウザではTLS 1.2以前は使えない(警告が出る)
  - TLS(SSL)の各バージョンのリリースと廃止勧告
    - SSL 3.0: 1995年リリース、2015年使用禁止勧告
    - TLS 1.0: 1999年リリース、2021年使用禁止勧告
    - TLS 1.1: 2006年リリース、2021年使用禁止勧告
    - TLS 1.2: 2009年リリース
    - TLS 1.3: 2018年リリース

接続時のの暗号スイートは安全ですか?(2/2)

- 「(互換性のために)仕様上は利用可能だが、実際は使ってはいけない」暗号スイートが存在したりする
  - 例: TLS 1.2以降でもTriple DESやRC4は仕様上は使えることになっているが、どちらも強度的にはダメダメ
- 例2: SSHの暗号鍵
  - 2020/5にOpenSSHはRSA公開鍵をデフォルトで無効化
    - 一応、鍵長4096bitならまだ安全だが、鍵長が長すぎて嬉しくない
    - 今新たに鍵ペアを使うならED25519が鍵長が短くて便利
      - OpenSSH 9.0から対量子として格子暗号+楕円曲線暗号のハイブリッドも利用可能になった
  - ただし、古い組み込み機器のSSHサーバが古い暗号スイートしか対応していないことも
    - 最新OS付属のSSHクライアントから、古い組み込み機器のSSHサーバに接続できない事例が増えているらしい
      - 「telnet復活」で対応した事例を聞くけど、個人的にはVMに古めのSSHが入ったOSを入れて対応するのがベストかと

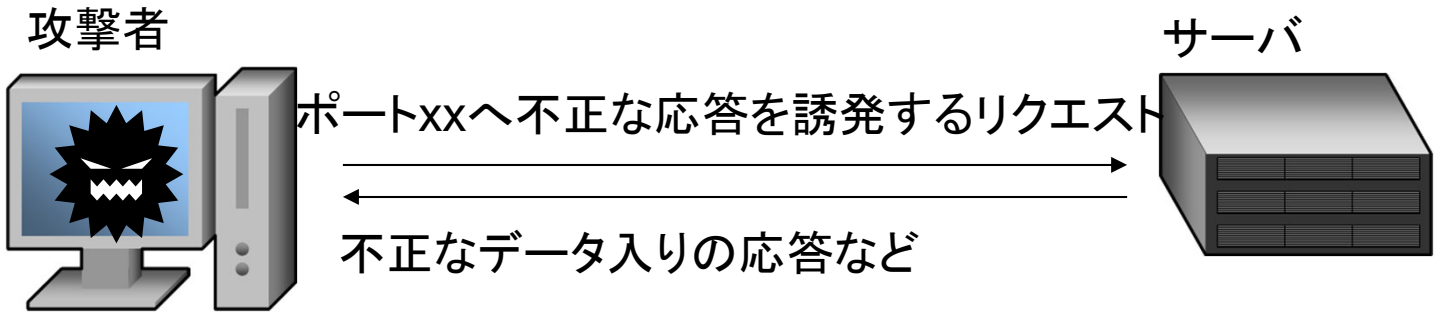
概要

- クライアント-サーバ間の通信路での攻撃
- 公開サーバー般に対する攻撃
  - ポート公開中のサービスの脆弱性を狙った攻撃
  - サービス不能(Denial of Service)攻撃
  - クライアント-サーバ接続のハイジャック
- Webサービスの提供者/利用者への攻撃
  - Webサービス提供者側への攻撃
  - Webサービス利用者側への攻撃
- 認証機構と突破の試み

公開中のポートに対する攻撃(1/3)

- 主なパターン
  - 公開中のポートに不正な応答を誘発するリクエスト
  - ポートを公開中のネットワークサービスを妨害するリクエスト
  - 認証を伴う公開中のポートに対して認証破り
- 接続を待ち受けているポートに接続できなければ、通信は開始しない

→ファイアウォールで不必要なポートへの接続要求をブロック



公開中のポートに対する攻撃(2/3)

- 一部OSで、ポート公開を伴う不要なネットワークサービスが動いていることはある
  - さらに、デフォルトのユーザ名とパスワードが利用可能だったりする
  - 最近は、OSやソフトウェア提供者側が学習してデフォルトをまともな設定にするようになった
  - と思ったら、最近は組み込み機器やIoT機器で問題が増加
    - さらに、機器性能から後述の古いプロトコルを利用している物が…

→空いているポートにデフォルトのユーザ名/パスワード

- SQLサーバのポートに対してSQLを送りつける物も多い
  - SQLサーバでリクエスト受理するIPアドレスの範囲を間違えたら…

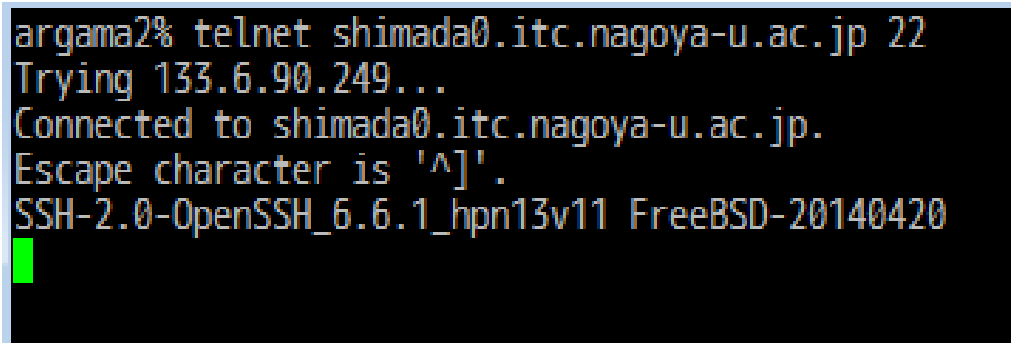
というか、適当にポートを空けていると色々なリクエストを試みてくる感じ

公開中のポートに対する攻撃(3/3)

- あまり古いサービスをインターネットに公開するのはよろしくない
- 例: 暗号化されていないパスワードがネット上を流れる
  - telnet, rsh → SSHが後継
  - FTP → FTPS, SSH(SSHサブセットのSCP, SFTP)が後継
  - POP3/IMAP → POP3S/IMAPSが後継
- 例: そもそも認証が緩すぎる
  - SMTP(クライアントとサーバの間において) →Submission
- その他、SMB、NFS(～v2)、Apple Filing Protocolなどのファイル共有系も弱い物が多い
  - SMB(v1)の脆弱性はWannCryに使われた点が記憶に新しい
  - AFPもApple自体が「インターネットに公開するな」と言っている

ポートの公開状態の調査(telnet)

- telnet IPアドレス/ホスト名 ポート番号
  - 最近のWindowsはtelnetを有効化しないと使えないので注意
- 応答を見ればポートで何のサービスが走っているか分かる
- 間にファイアウォールが入っている時の挙動が2種類ある
  - RSTを返す →即connection refused
  - SYN/ACKを返さない →応答が返ってこない



ポートの公開状態の調査(nmap)

- 全ポートを調査するツールの1つ
  - オプション次第では応答からの情報も提示
- 攻撃(の前の事前調査)と間違えられないように注意

```
argama2% nmap shimada0.itc.nagoya-u.ac.jp

Starting Nmap 7.12 ( https://nmap.org ) at 2016-05-17 10:28 JST
Nmap scan report for shimada0.itc.nagoya-u.ac.jp (133.6.90.249)
Host is up (0.013s latency).
rDNS record for 133.6.90.249: gerbera
Not shown: 946 closed ports, 50 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
2000/tcp open  cisco-sccp
5060/tcp open  sip

Nmap done: 1 IP address (1 host up) scanned in 8.74 seconds
argama2% telnet shimada0.itc.nagoya-u.ac.jp 2000
Trying 133.6.90.249...
Connected to shimada0.itc.nagoya-u.ac.jp.
Escape character is '^]'.
Connection closed by foreign host.
argama2% telnet shimada0.itc.nagoya-u.ac.jp 5060
Trying 133.6.90.249...
Connected to shimada0.itc.nagoya-u.ac.jp.
Escape character is '^]'.
Connection closed by foreign host.
argama2%
```

```
argama2% nmap -A -T4  shimada0.itc.nagoya-u.ac.jp

Starting Nmap 7.12 ( https://nmap.org ) at 2016-05-17
Nmap scan report for shimada0.itc.nagoya-u.ac.jp (133
Host is up (0.017s latency).
rDNS record for 133.6.90.249: gerbera
Not shown: 950 closed ports, 46 filtered ports
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 6.6.1_hpn13v11 (Fre
)
| ssh-hostkey:
|   1024 0e:4a:56:18:f1:0e:26:dc:20:5f:e9:97:32:ff:fe
|   2048 be:ac:47:5c:67:0d:be:83:90:6a:e5:1d:b9:ef:9d
|_  256 1b:dd:00:93:0b:21:87:d4:c2:5d:b6:84:a8:51:e9:
80/tcp   open  http       Apache httpd 2.4.20 ((FreeB
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.20 (FreeBSD)
|_http-title: Tentative transfer
2000/tcp open  tcpwrapped
5060/tcp open  tcpwrapped
Service Info: OS: FreeBSD; CPE: cpe:/o:freebsd:freebs

Service detection performed. Please report any incorr
.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 25.77
argama2%
```

ポート公開中のホストを探すサービス

- ネット上をクロールしてポート公開中のホストを確認してまとめてくれるサイトが複数ある
- 中には、わざわざその公開ポートから確認できるデータの一部もまとめてくれている所も
  - サーバソフトウェアのバージョンや受け付ける暗号スイートなど
- 自組織が運用中のホストの穴のチェックに活用できる
- 代表例
  - Shodan
  - Censis

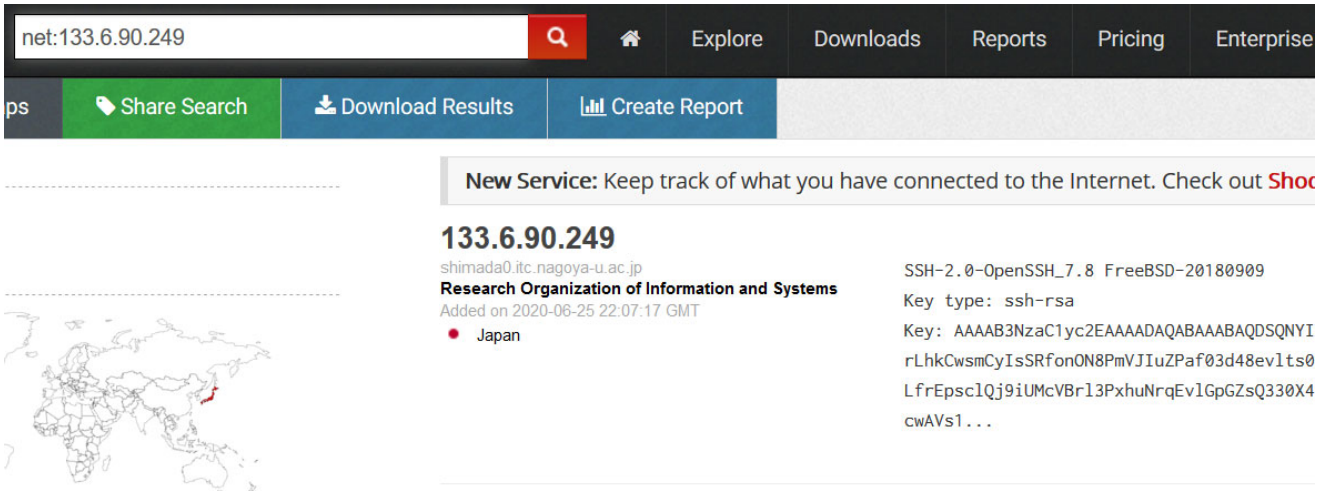
Shodan

https://www.shodan.io/

- ポートの公開状態を検索できるサイトの草分け
- Webcam、BEMS(Building Energy Management System)を始めとする各種EMS、各種IoTデバイス
- ac.jpドメインのメールアドレスでユーザ登録するとアカデミックな利用権限になる

Shodanで嶋田サーバを検索した結果

- IPアドレスの指定は「net:*IPアドレス*」の形式で
  - CDIR形式で範囲しても可能(例: net:192.168.0.0/24)
- SSHサーバ動作中ということがバージョンや公開鍵を含めて分かる
- Webサーバを動かしていると、Webサーバソフトウェアとか HTTPSの公開鍵とかも表示される



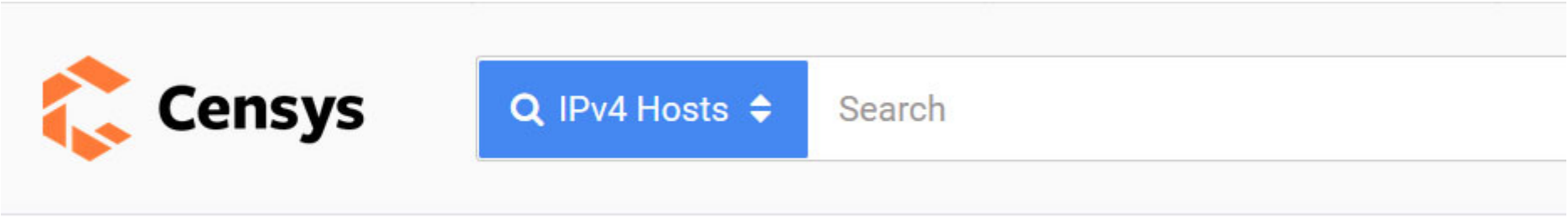
Shodanでいろいろ検索してみよう

- 動作中サービスに関連して収集された情報をキーワードに検索可能
  - 例: OpenSSH, FreeBSD
- 「hostname:*example.com*」「country:*国名*」「org:*ネットワーク運用組織名(ISPなど)*」な検索可能
- 複数出てくる場合、右のように統計情報の形でのサマライズも出る
- スクレイピングなどをやりたければPythonにshodanライブラリあり
- 詳細はhttps://help.shodan.io/

Censis

https://www.censys.io/

- Shodanよりも検索能力が強化されているのが売り
  - 登録されているデータに対してGoogleと同程度の検索を可能
    - OS名、HTTPサーバソフトウェア名、サーバが返すHTTPヘッダ、など
  - 正規表現でのマッチ可能
    - https://censys.io/certificates/help
- ミシガン大学の研究者が開設



Censysで嶋田サーバを検索した結果

- SSHの受付暗号スイートの提示など、Shodanより細かい

133.6.90.249 (shimada0.itc.nagoya-u.ac.jp)

Summary | WHOIS | Raw Data

Basic Information

OS	FreeBSD
Network	SINET-AS Research Organization of Information and Systems, National Institute of Informatics (JP)
Routing	133.6.0.0/16 via AS11537 , AS2907
Protocols	22/SSH
Tags	SSH

22 / SSH

SSHv2 Handshake DETAILS

Server	OpenSSH 7.8
Banner	SSH-2.0-OpenSSH_7.8 FreeBSD-20180909

Host Key

Algorithm	ecdsa-sha2-nistp256
Fingerprint	0004e8dce17133a44f9b9eec17708469b4b25c99d00cd14dfd7bfab091fff1d1

Negotiated Algorithm

Key Exchange	curve25519-sha256@libssh.org
Symmetric Cipher	aes128-ctr [↑] | aes128-ctr [↓]
MAC	hmac-sha2-256 [↑] | hmac-sha2-256 [↓]

Geographic Location

Country	Japan (JP)
Lat/Long	35.69, 139.69
Timezone	Asia/Tokyo

Censisで色々と検索してみよう

- "192.168.0.0/24 and FreeBSD"
  - 192.168.0.0/24でOSがFreeBSDなホストが見つかる
- "192.168.0.0/24 and SSH"
  - 192.168.0.0/24でSSHが開いているホストが見つかる
- "192.168.0.0/24 and Apache and 2.4"
  - 192.168.0.0/24でApache 2.4が動いている
- ポート番号を変更しても見つけてくれます(ついでにタグ付けしてくれる)
  - http: 9000, 8443, 8172, 4004, 20000, など
  - ssh: 6478, 60022, 60122, 60222, 44413, 2222, 22222, 20100, など
- 詳細はhttps://censys.io/ipv4/help

公開ポート関係 その他もろもろ

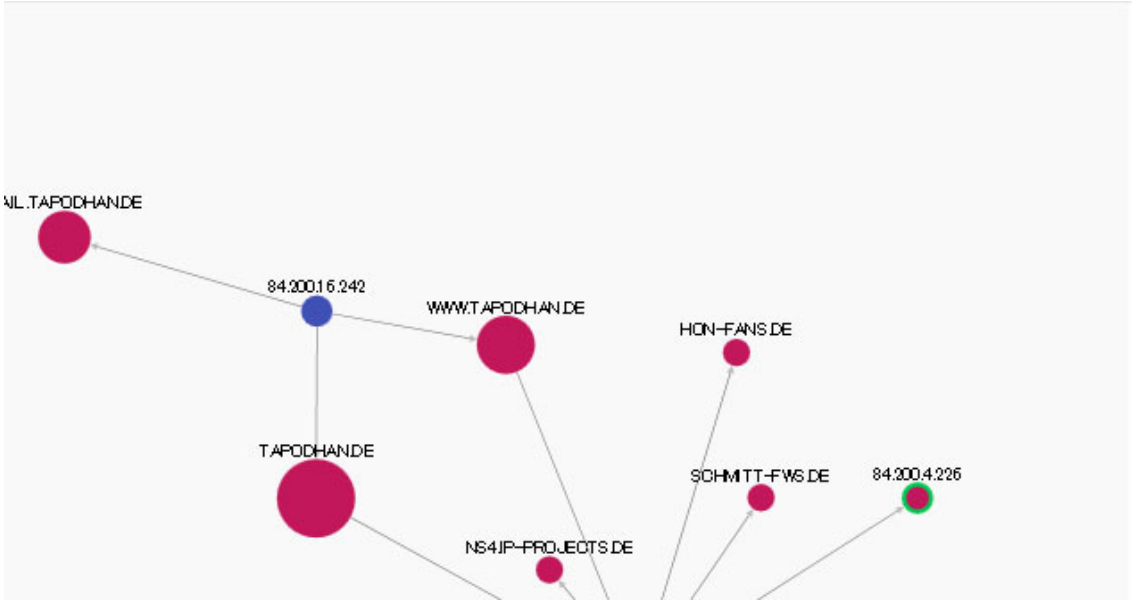
- Inscam[1]
  - Webcamに特化した公開ポートリストアップサイト
  - セキュリティの甘いIPカメラの画像をリアルタイムに掲載
    - 初期パスワードのまま、パスワードが空、など
- zmapやmasscan
  - 自組織所有のIPアドレスを調査したい場合に使えるツール
- 他にもZoomEye, FOFA, ONYPHEなどがある

[1] http://nlab.itmedia.co.jp/nl/articles/1601/21/news137.html

ThreatCloud

- 通信元IPアドレスか怪しいかどうか提示するサービス
  - 他に関連する通信先も提示してくれる
- 例: ランサムウェアPetyaの通信先IPアドレス
- 他にも、CYMON(https://cymon.io/)などいくつかサービスがある

Petyaの通信先IPアドレスをThreatCouldに入れた結果



IP > 84.200.16.242

Most users have voted this as MALICIOUS

Is this malicious? Yes No

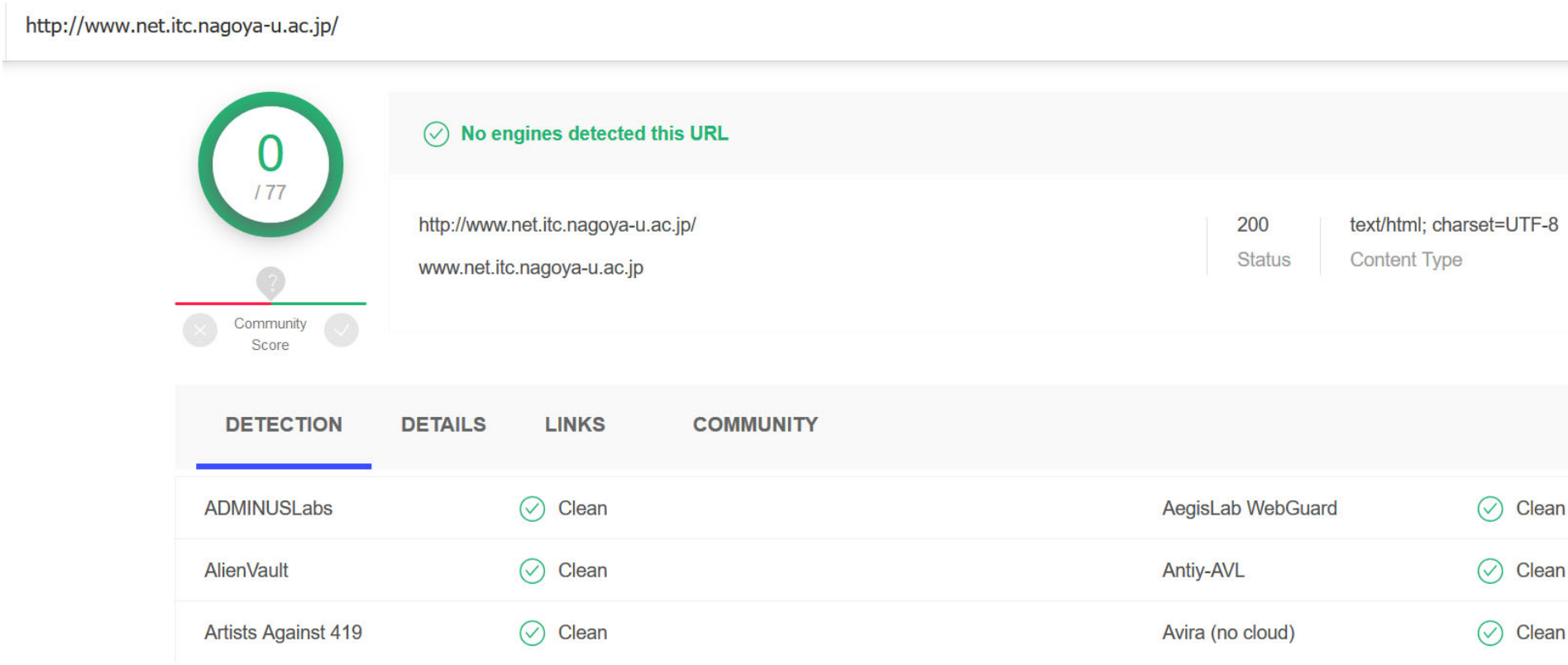
More information on this IP is in **AlienVault OTX**

IP WHOIS

Property	Value
Country	Germany

VirusTotalのURLレピュテーション機能

- そのURL(の下)に悪意のあるコンテンツがあるか(あったか)どうか、複数のアンチウィルスエンジンでの結果を提示



公開ポートへの不正な応答を誘発させるリクエストの事例

基本的に脆弱性として報告されているが…

- サーバの動作が止まる
  - 例: BIND(DNSサーバ)でサーバ停止の脆弱性(CVE-2016-2848)
- パケットが処理できなくてDenial of Service状態になる
  - 例: 組み込みシステム用シーケンサでリソース枯渇の脆弱性(CVE-2020-13238)
- 任意のコードが実行される
  - 例: EternalBlueと呼ばれるSMB v1の脆弱性(CVE-2017-0144)
- 秘匿すべき情報が応答に入る
  - 例: フロントパネルのパスコードを返す産業用プリンタ(CVE-2019-10960)

興味がある人はCVE DBなどでmalicious packet/requestなどで検索してみよう

公開中のポートへの攻撃観測

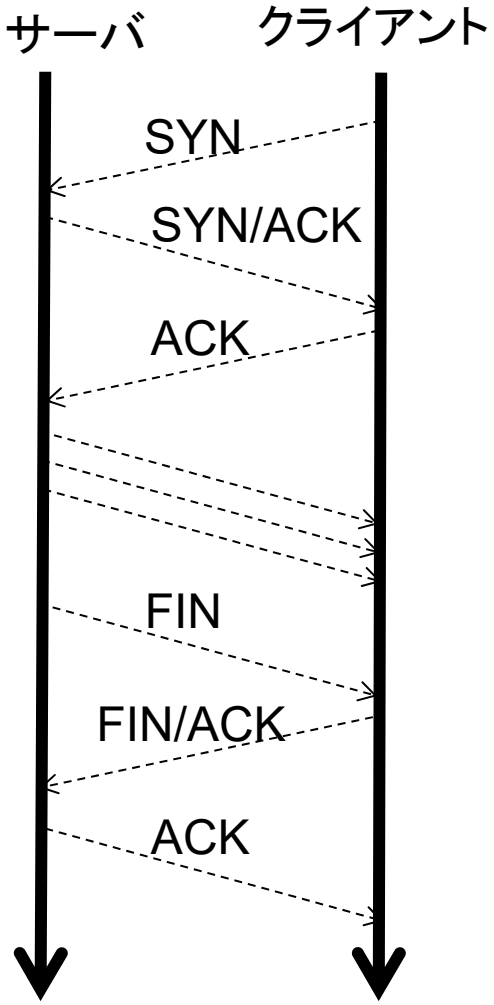
- ハニーポットという、わざと開いているポートを設定して攻撃観測するサーバやサーバソフトウェアが存在
  - 低対話型ハニーポット: 空けたポートへのリクエストのみを収集
  - 高対話型ハニーポット: 空けたポートへのリクエストに対して応答を返して接続者の挙動をより深く観察
    - 攻撃者がシステムに侵入してバックドア設置や配布用マルウェア設置まで観察することも
  - 色々なハニーポットフレームワークがあるので、(個人で)試してみるのも面白い
    - 最近、明らかな攻撃観測やってると腹いせ攻撃が来るらしいので、やるならクラウドサービスとかで
- もっと大規模な物では、未割り当てIPアドレス領域(ダークネット)へのリクエストをまとめて観測することも
  - 余談: 最近、ダークWebとダークネットをごっちゃにされていることが多くて嫌な感じ

サービス不能攻撃(DoS攻撃)

- Denial of Service(DoS)攻撃
  - サーバやネットワークの処理能力を超えた大量のリクエストを送りつけて、処理不能にさせる
  - 攻撃元を分散させるとDDoS(Distributed DoS)攻撃
  - アクセス集中で意図しないDoS攻撃状態になることも
- 様々な階層で実行可能
  - SYN(/FIN/ACK) flood(L4)
  - UDP Flood(L4)
    - 最近ニュースとかで話題になるDoS攻撃はほとんどこれ
    - DNSリフレクションなどのリフレクション系攻撃が問題を大きくしている
  - HTTP GET flood(L5)
  - Slow HTTP DoS(L5)
- ちょっと変わったDoS

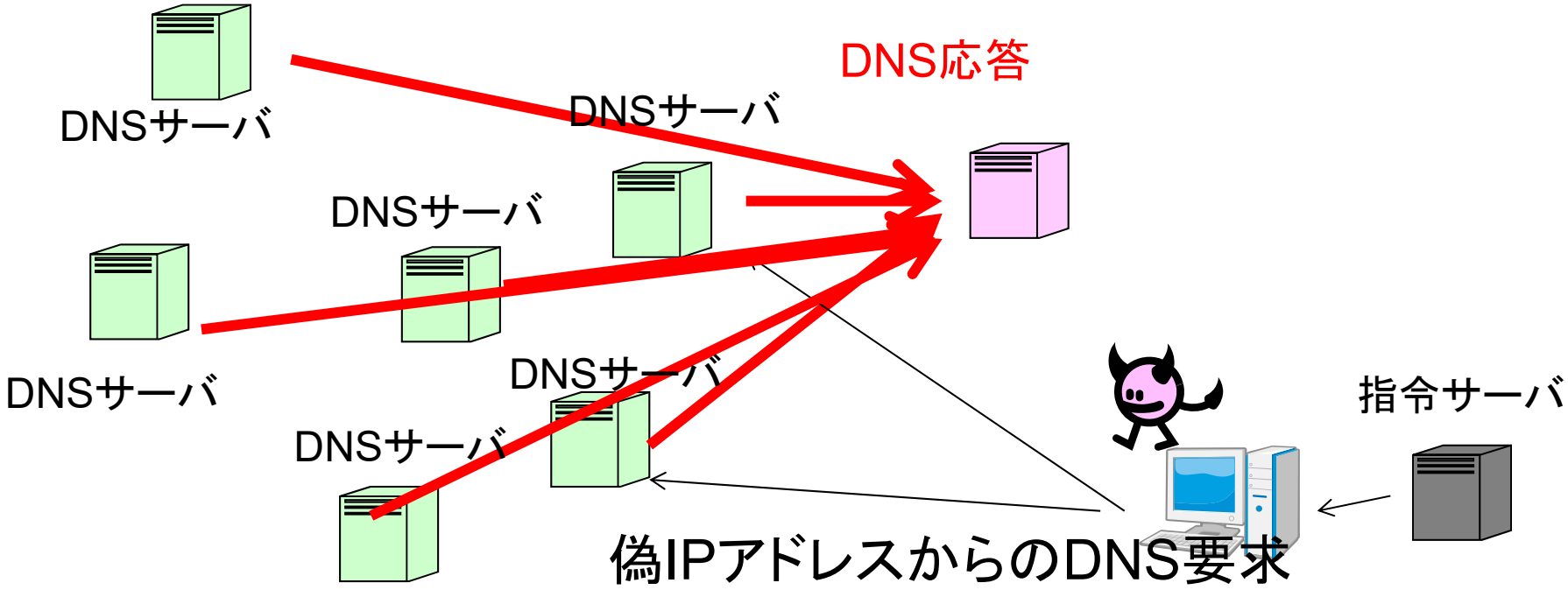
SYN(/FIN/ACK) Flood

- TCPは3ウェイハンドシェイクで通信の開始/終了
- この最初のSYNだけを送りまくる攻撃
  - サーバ側は新TCPセッションの準備ために計算機資源を消費
    - 厳密には、発信元を偽装IPアドレスに設定したり
  - FIN/ACKなどを送る版もある
- 対策
  - ネットワークスイッチによるフィルタリング
    - サブネットの入り口で偽装IPアドレス(ありえないパターン)のパケットを落とす
  - サーバ側ではSYN cacheなどでSYNを一旦待たせるなど



UDP Flood(1/2)

- 大抵はリフレクション攻撃も併用(UDP送りつけだけは稀)
  - いくつかのUDPプロトコルにおいてリクエストに対して応答の方がサイズが大きいことを利用
  - 要求の方は短い問い合わせでも、応答の方は多くのデータがあるので、データサイズは「応答>要求」



UDP Flood(2/2)

- 防衛の面では、UDPは送信/受信ともに同じポートを使うのがやっかい
  - 例: NTPリクエストはNTPサーバのUDP 123ポートへ送信、応答はクライアントのUDP 123ポートへ返信
  - (攻撃の)応答なのか正規の問い合わせの応答なのか分かりにくい
- DNS/NTPのリフレクション攻撃が有名
- マイナーなUDPプロトコルにも攻撃の手は伸びる
  - SSDP(Simple Service Discovery Protocol), chargen(CHARcter GENerator), SNMP(Simple Network Management Protocol)など
  - 名大では外部にポート公開することは禁止されている

UDP Flood系の増幅率

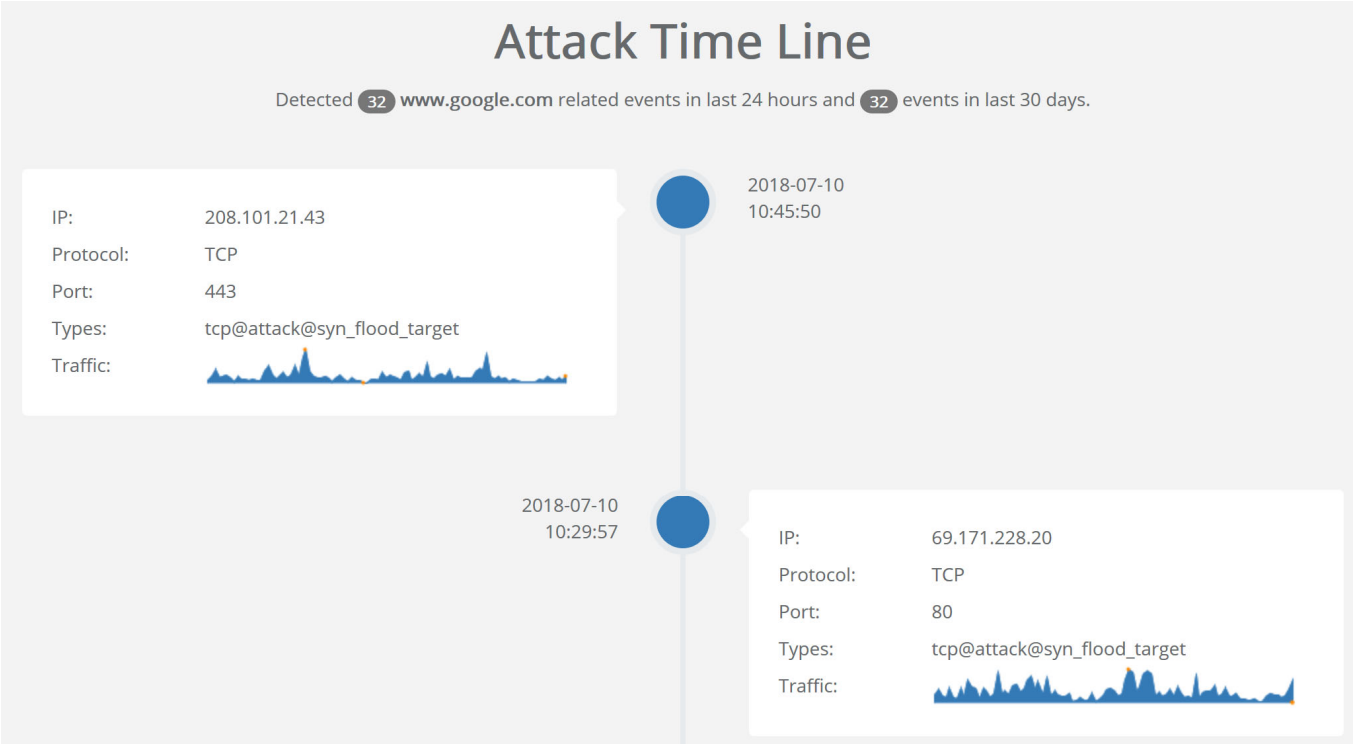
とりあえず、できるだけ多くのデータを送る応答を選択した時

- DNS(UDP 53): 8倍
- TFTP(UDP 69): 60倍
  - 古い組み込み機器で使われる非常にシンプルなFTP(ファームウェア更新など)
- NTP(UDP 123): 19-206倍
- SNMP(UDP 161): 650倍
  - ネットワーク機器のステータスなどのログ報告
- SSDP(UDP 1900): ?
  - Universal Plug and Playに必要なデータを送る
- chargen(UDP 19): 無限大?
  - 動作確認などを目的としてひたすら文字列を生成するサービス

DDoS Mon: DDoSの様子を見れるサイト

https://ddosmon.net/

- IPアドレスもしくはドメイン単位でDDoSの履歴を見ることが可能
  - もちろん、(CDNなどの)観測ポイントを経由している通信のみ



HTTP GET/POST Flood

- HTTP GET要求を多数送る
  - 要求よりも応答のデータが特に多いと発生しやすい
    - 動的にWebページを生成してたりするとさらに発生しやすい
  - Webブラウザのリロードで意図せずに発生することも
    - Webページが重くてなかなかデータが来ない
      →とりあえずリロードしてみる
  - 対策
    - コンテンツキャッシュサーバの準備
      - 最近だとContent Delivery Network(CDN)に任せる方が手軽
    - 単位IPアドレスあたりのセッション数の制限
- HTTP POST要求を多数送る
  - Webサーバ側でPOSTされたデータの処理が重いと発生しやすい
  - 対策(上とは異なるもの)
    - POSTを送る前に別ページでセッションID(Cookie)を必要とさせる

Slow HTTP DoS

- HTTPセッションが途切れない程度に通信を継続
- 被害
  - セッション維持のためにHTTPサーバにおけるメモリリソース等の浪費
  - TCPポート数が不足することによるサービス不能
- 対策
  - タイムアウト時間の短縮(ただし、ユーザから不満が出ることも)
  - 単位IPアドレスあたりのセッション数の削減
    - ただ、最近のクライアントは数十セッションを張って高速化したりするので、そういう利用で遅くなる弊害(サービス悪化)
    - 参考: Google Mapは利用可能セッション数10以下になると目に見えて表示が遅くなる
    - 参考: クライアントの多セッション利用は大規模NATにおける1グローバルIPアドレスあたりの収容数にも影響する

他のDoS攻撃

- ICMP Flood
  - ping(ICMP ECHO)を送りまくる
  - パケットサイズが大きいpingを送ることも
    - 変なペイロードを載せたpingという話もちょくちょく聞く
- アプリケーションなどの実装の脆弱性を利用したDoS攻撃
  - 規格外のサイズのパケットを送ると処理が止まる、規格外のデータを送ると処理が止まる、など
  - 意図せず実装の脆弱性をついて攻撃に間違われた事例も
    - 岡崎市立図書館事件: 蔵書検索システムが検索ごとに消費したリソースを解放しない

      →ゆっくりした頻度のクロールでもリソース不足へ
- (システムトラブル時にヘルプデスクへの電話が集中するDoS)

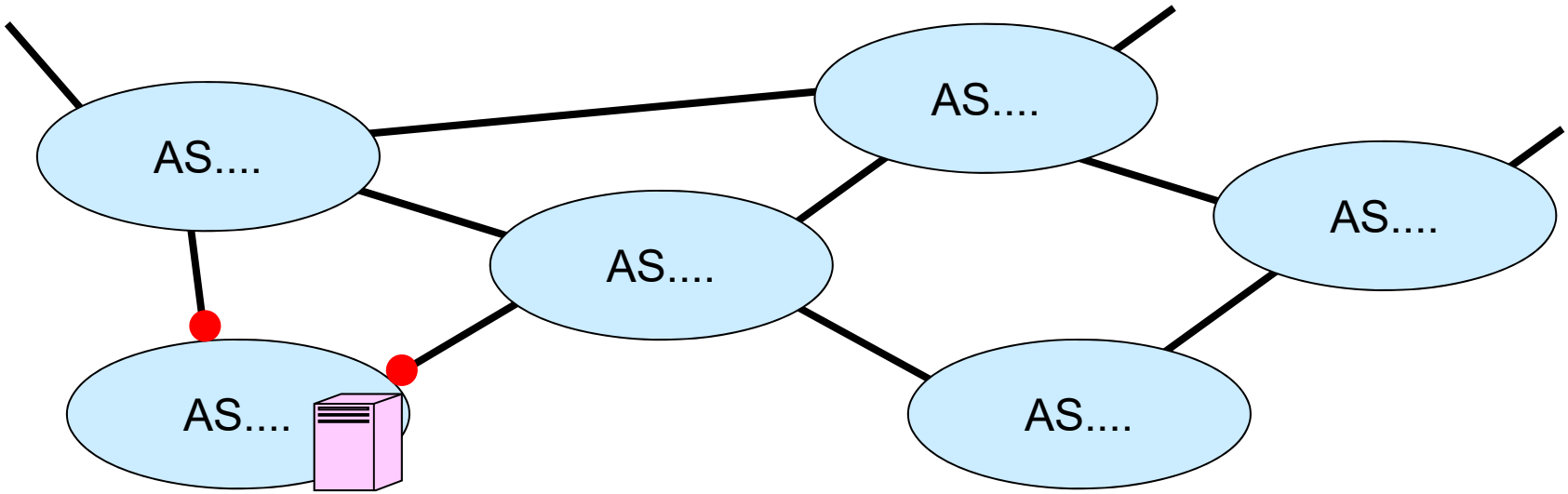
UDP Flood対策

- 攻撃参加しないように
  - サーバに対して要求に応答する範囲を制限する
  - 対外接続ネットワークスイッチでEgressフィルタリングをする
    - 自組織以外のIPアドレスをsrcとするものを落とす(偽装IPアドレス)
    - 組織のセキュリティ・ポリシで許可した以外のアウトバウンド通信を禁止
- 防衛
  - 基本的に完全な防衛は難しい
    - 「全く無意味なパケット」でも物理層を埋めることはできる
  - いろいろと対策はあるが、「何かを切り捨てる」必要は出てくる
    - ある国からの正規のアクセスは(一時的に)切り捨てる
    - 自組織外からのアクセスは(一時的に)切り捨てる

UDP Flood対策(自ネットワーク入口)

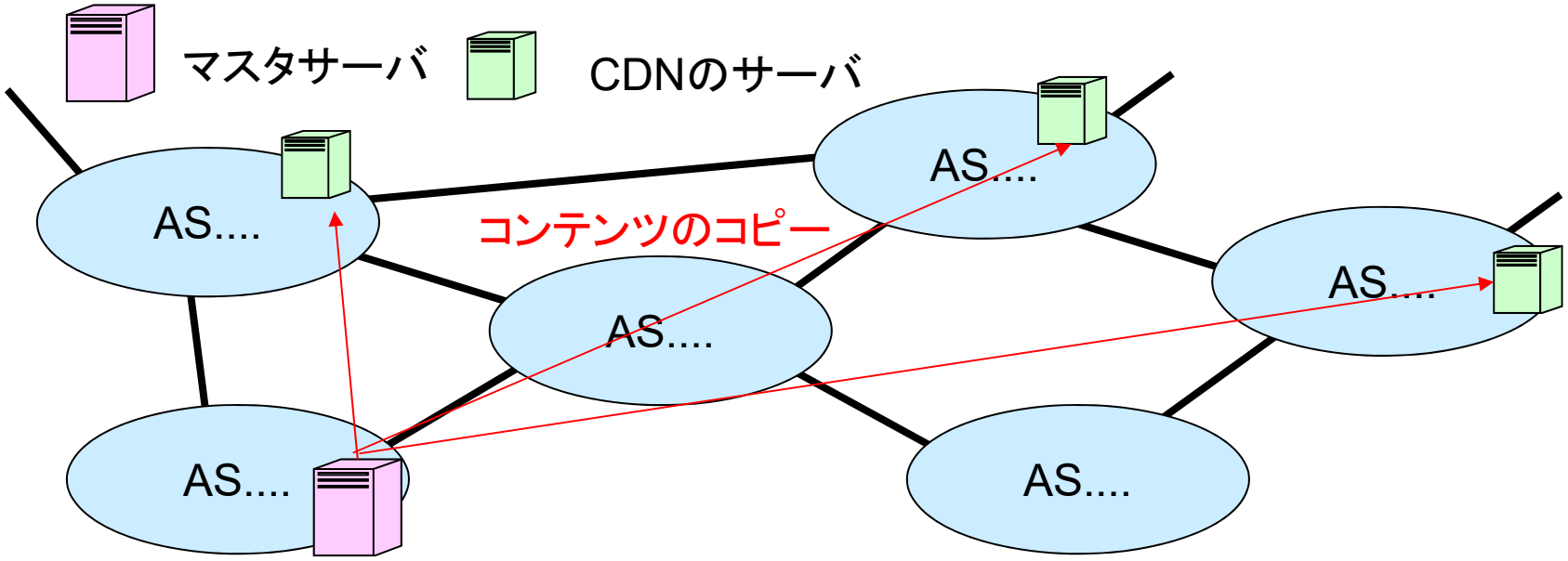
- 外から来ることが考えられないプロトコルは落とす
  - ステートフル・ファイアウォールを導入してLANから出してない要求への応答は落とす
- 対外接続部で通信を制限する

→対外接続の帯域を食われることには変わらないが、サーバや内部からのユーザは対応できる



UDP Flood対策(自ネットワーク上流) (1/2)

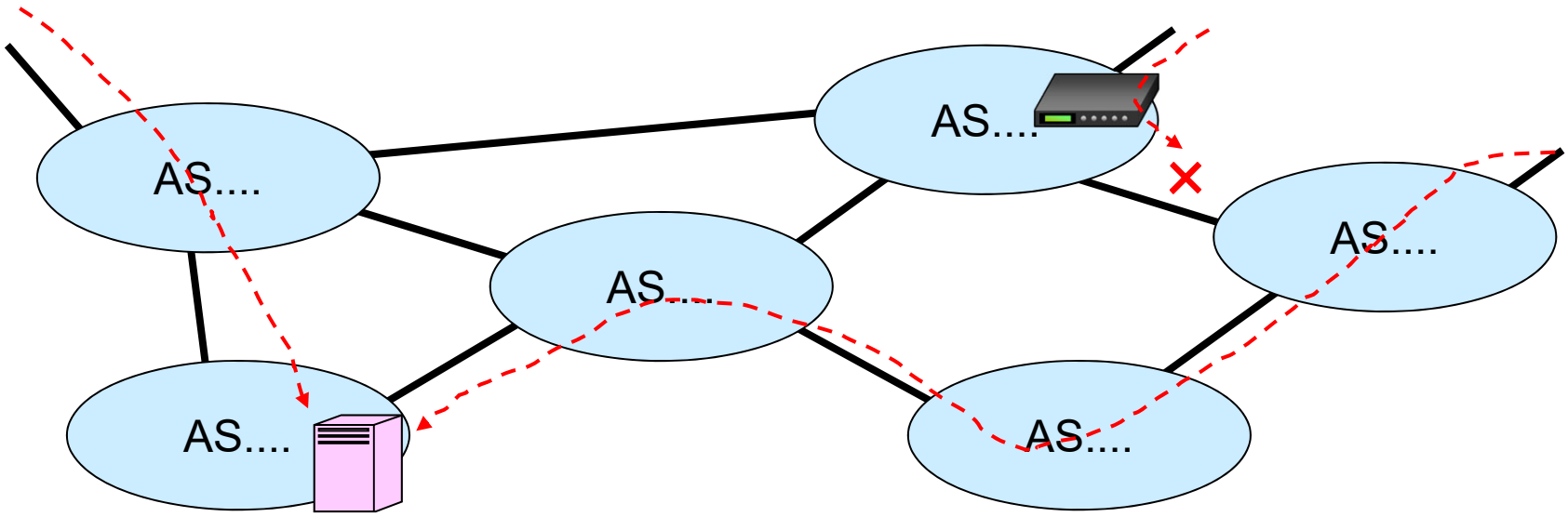
- Content Delivery Network(CDN)を使って耐える
  - CDNはユーザに近い所にコピーコンテンツを保持して応答
    - DNSを引くと近いCDNのサーバが返ってくる
  - マスタサーバの負荷低減にもなる
  - ただし、お金はかかる(DoSオプションなどもある)
  - マスタサーバ自体を直接狙われることがある



UDP Flood対策(自ネットワーク上流) (2/2)

- BGPで特定の通信を捨てさせる指示が可能(Blackhole routingの応用)
- BGPのオプションで、特定のASに対してのみ捨てさせるようBGPの広報を行なう

→特定の国などからの流入を防ぐ



通信量ベースのDoSについてもっと調べたい方

- Akamaiのレポートが詳しい
  - Akamai: Content Delivery Network(CDN)の最大手
    - 提供するコンテンツをインターネット上のいくつかの場所にキャッシュ可能とし、本サーバの負荷を減らす
  - https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report/
  - 4半期ごとにDoS攻撃の傾向をレポート掲載
    - 攻撃の変遷も見ることが出来る
    - リフレクション攻撃に使われるプロトコルの変遷が面白い
- 2020年にはついに2TbpsオーバーのDDoSが観測される
  - 4,5年前にマルウェアMiraiによって500Gbps級のDDoSが話題になったばかりなのに…
  - 世界の通信量の伸び(年率23%、前資料)よりも早いペースの伸び

セキュリティログへのDoS

- セキュリティに関係するログは数ヶ月から数年残すべきもの
  - セキュリティインシデント後のフォレンジック対応のため
  - 例: IDS、ファイアウォール、Web proxy、DHCP、認証
- このログを溢れさせたり欠落させることを目的としたDoSが行われることもある
  - 他の攻撃をログ上で隠蔽するためにDoS
  - サーバへのDoS攻撃のあおりを食らってログ側がやられることも
  - そもそも、ロギングの設定が悪くて、通常の偵察行動とかでセルフDoSに陥ることをやらかすことも
- SNMPなど、UDPベースでログをログサーバに送るバックエンドだと、サーバ性能によって容易にログ欠落することも

無線LANへのDeauthentication AttackによるDoS

- 無線LAN(IEEE 802.11)のDeauthenticationフレームは単純過ぎる
  - MAC header(L2) + Deauthentication通知
  - 当然、L2レベルなので暗号化しようがない
- MACアドレスを偽ってDeauthenticationを送るDoS攻撃が成立する
  - 偽無線LANアクセスポイント(Rogue AP)に接続替えさせるとか、他の攻撃の起点にも利用される
- 有料無線LANに誘導するために、某ホテル大手が持ち込みWiFiルータ等にDeauthenticationを送っていた話も[1]
  - FCCにバレて60万ドルの罰金

(*1) https://www.cnn.co.jp/business/35054743.html

概要

- クライアント-サーバ間の通信路での攻撃
- 公開サーバー般に対する攻撃
  - ポート公開中のサービスの脆弱性を狙った攻撃
  - サービス不能(Denial of Service)攻撃
- Webサービスの提供者/利用者への攻撃
  - Webサービス提供者側への攻撃
  - Webサービス利用者側への攻撃
- 認証機構と突破の試み

どのように攻撃されることが多いか?

- ユーザからの入力に応じて処理結果を変更する構造に攻撃されることが多い
  - ユーザからの入力をWebページに表示
  - ユーザからの入力に応じてデータベースを検索
  - 「ユーザの入力が入る所(URLクエリ、HTTPヘッダなど)に攻撃者側不正な値を入れる」→「提供者/利用者側に攻撃成立」のパターン
- Webサービス提供者側への攻撃
  - その入力を表示するとスクリプト等が実行される
  - その入力をデータベースに入れると想定外の処理される
- Webサービス利用者側への攻撃
  - (攻撃者が事前に細工したURLやWebページ閲覧で)
    - Cookie等の窃取や(次の攻撃を目的とした)Cookie設定がされる
    - 正規のコンテンツに悪意のあるコンテンツを混入させる

クロスサイトスクリプティング(1/4)

- URLクエリやHTTP POSTでフォームへの特定の応答により、攻撃者が用意したスクリプト(プログラム)が実行される
  - フォームに入力された値やURLクエリの値を表示させる時のミス
- 基本的に、「入力の一部でタグを途中で終了させ、その後にスクリプトを挿入」の形
  - 例: http://..../hoge.cgi?initdata=hogeと入力すると<input initdata="hoge">となるHTMLを動的生成におけるJavaScript挿入
  - 入力例: http://..../hoge.cgi?initdata="><script>alart(1);</script>"<input initdata="
  - 出力例: <input initdata=""><script>alart(1);</script>"<input initdata="">
- 基本的な対策: 入力データをエスケープ
  - < → < > →> " → " ' → $#39; & → &

クロスサイトスクリプティング(2/4)

- 略称はXSS
  - Web関係でCSSだとCascading Style Sheetが先にあったため
- 攻撃者が悪意のあるスクリプト(通常はブラウザ上で実行可能なJavaScript)を構成したりして悪用
- 攻撃者による悪用時の被害例
  - 偽入力ポップアップを作成してパスワードや個人情報を窃取
  - 本物コンテンツの上に偽コンテンツを表示してクリック位置等を誤認させて誤操作させる
  - 保存されているCookieの窃取
  - Cookieの新規設定
    - セッションID固定攻撃(既存のセッションIDがCookieにあるとその値を利用する脆弱な設計)の布石

クロスサイトスクリプティング(3/4)

- 攻撃者側もあの手この手でスクリプトを動かす試みをしてくる
  - 複数の入力文字列を連結するとスクリプトが生成される
    - 例: 氏名を別々の欄に入力するフォームで、連結すると動くスクリプトを分割入力
  - 怪しい文字列を置換したり削除したりするとスクリプトが生成される
    - 例:「scriscriptpt」のように、scriptという文字を削除すると動くスクリプトが生成されるように
- Microsoft EdgeなどがXSSフィルタを搭載ブラウザもある
  - FirefoxではNoScriptアドオンとかで追加可能
  - 誤検出することもあるし、そもそも攻撃者はフィルタにひっかからないか事前チェックできる
  - 今度はXSSフィルタ悪用する攻撃(→一部を削除するとスクリプトとして動く)も出てきたり…

    →EdgeからXSSフィルタを削除することも検討されたりする

クロスサイトスクリプティング(4/4)

- クロスサイトスクリプティングを起点として、多種多様な攻撃を実現することが多い
  - ユーザ側の閲覧データ加工: 強制ブラウズ、リモートファイルインクルード
  - Man-in-the-Middleによる(セッション)ハイジャック: セッションID固定攻撃、HTTPヘッダインジェクション、(iframeなど上位のページを設定しての)Cookieの読み出し
  - その他: オープンリダイレクタを利用した誘導
  - サーバのデータ閲覧など: SQLインジェクション、LDAPインジェクション、ディレクトリトラバーサル、OSコマンドインジェクション、Xpathインジェクション、メモリ初期化ミスを利用したメモリリーク

iframeとXSS併用による悪意のあるページの埋め込み

- iframe(inline frame): ページの一部にフレームを設定して、別HTMLを表示するフレームワーク
  - 古来のframeと比較して、フレームの境目が分かりにくい
- XSS系の攻撃においてよく悪用される
  - 攻撃(詐欺)ページの中にiframeで元のページを表示する
  - 元ページの上にXSSで背景を塗りつぶした攻撃ページのiframeを置いて、元ページを隠蔽
  - 攻撃JavaScriptを動作させるために透明もしくはサイズ極小のiframeを設定して、その中で攻撃JavaScriptを動作させるHTMLを表示

サニタイジング(sanitizing)

- sanitize: 衛生的にする、消毒する、「無害にする」
- (主にWebアプリケーションで)入力データにおいて「そのまま処理すると有害」な部分を無害化する
  - プログラミングにおけるエスケープ処理と似た感じ
- サニタイジングしない…
  - HTMLのタグを無理矢理閉じた後に、悪意のあるSQL、JavaScriptなどを挿入される
  - OSコマンドへの引数の後に区切り文字を入れた後に別コマンド
- よくやるサニタイジング
  - HTMLのタグで使われる文字の処理: < → < > →> " → " ' → $#39; & → &
    - XSSやSQLインジェクション対策にもなる
  - その他の各種実行系(シェルなど)での区切り文字

クロスサイトリクエスト偽造(CSRF: Cross Site Request Forgery)

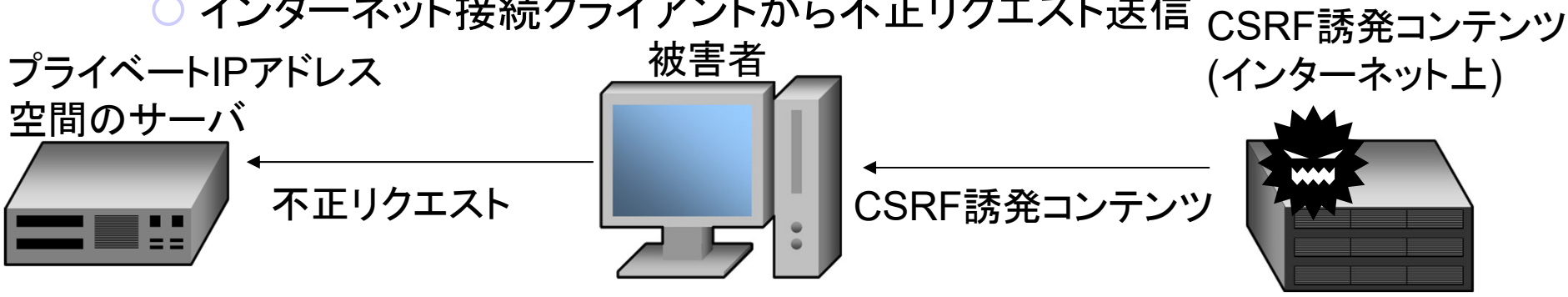
- ここでのリクエスト = HTTP POSTなどによるクライアントからサーバへのリクエスト
  - XSSはブラウザ内で完結するが、CSRFは外部に出力
- 「偽のリクエストを作り、サーバ側のデータを書き換えたりできる」のがこの攻撃の特徴
  - 認証のあるページでも、ユーザが認証終了した後(認証済みCookieを持っているとか)していたら書き換え可能
  - 例: パスワード変更ページに攻撃者側があらかじめ設定したパスワードに変更
  - 例: 掲示板への悪意のある書き込み
- 対策
  - データ更新ページの遷移前後で追加のセッションキーを設定
  - データ更新承認時に再度認証を要求する

サーバサイドリクエスト偽造(SSRF: Server Side Request Forgery)

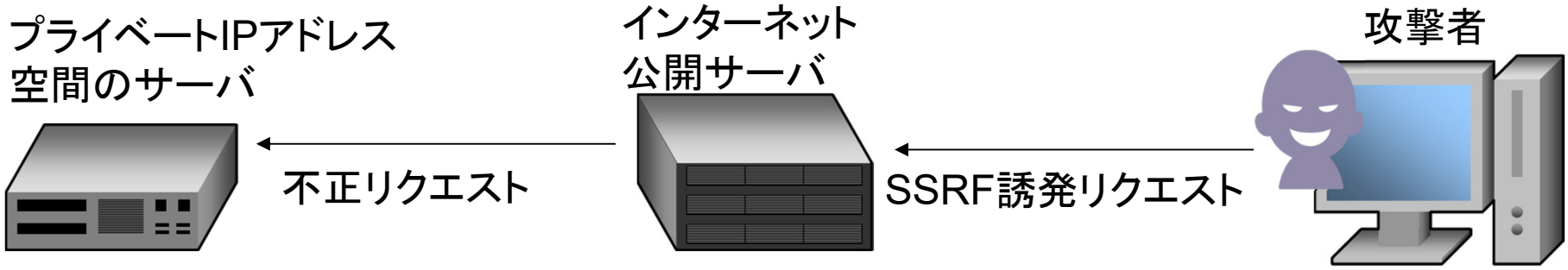
- Webサーバ側から別WebページにGET/POSTなどのリクエストを強制的に出させる
- 例: Webページプレビュー機能があるWebアプリケーションにおいて、プレビュー先をURLクエリ/POSTで指定可能
  - CMSなどで編集後のページのプレビューで、テンポラリのページを生成してそのURLを返すような実装とかで起こりうる
- CSRFと同様に外部サービスへ影響を及ぼすことも可能
  - 例: Web掲示板に書き込みリクエストを出す
- Webサーバが組織内専用ネットワークのIPアドレスも持つ場合、組織内専用サーバにリクエストを出すことも
  - 例: プライベートIPアドレスを適当に埋め込んだURLを作ったら組織内専用Webページらしきものが見えた

リクエスト偽造とプライベートIPアドレス空間

- 個人的には、プライベートIPアドレス空間のサーバへのCSRF/SSRFが怖い
  - 同じプライベートIPアドレス空間につながっているインターネット接続端末などからCSRF/SSRFで不正リクエストを送りこまれる
  - インターネット接続クライアントから不正リクエスト送信



  - インターネット公開サーバ経由で不正リクエスト送信



SQLインジェクション(1/2)

- ユーザからの入力をもとにデータベースへのSQL(Structured Query Language)司令文の作成において発生
- SQLを使って認証処理を同時にやっていたりすることもある
  - 例: ユーザ名とパスワードを問い合わせるSQL(条件文)を作り、真が返って来たら認証成功
- 例: ログイン処理でフォームのユーザ名とパスワードが一致するか?
  - フォームの入力はinuserとinpasswdとする
  - 条件文例: user = 'inuser' AND passwd = 'inpasswd'
  - 何も対策されていないと、inpasswdに「a' OR 'a' = 'a」を設定する

→user = 'inuser' AND passwd = 'a' OR 'a' = 'a'

- ORに恒真節が入って常に真が返る →認証突破

SQLインジェクション(2/2)

- 「データベースの全データをダンプ」というSQLを構成して結果をブラウザに表示させることができると大被害
- DBのテーブルにサーバ内のファイルを指定して、サーバ内の別データからの情報流出につながることも
- データ管理の簡略化や、Webアプリケーション高機能化のためにDBが動いている事例は多い
- 基本的な対策: サニタイジング、値の範囲のチェック

HTTPヘッダインジェクション

- ユーザの入力を応答のHTTPヘッダの一部に取り込む構成において発生
- 特に改行が処理されていないと怖い
  - 改行した後に別のHTTPヘッダ要素を自由に作成できる
  - 偽のCookieを発行することができてしまう
  - Locationヘッダを生成して別サイトへの転送ができてしまう

オープンリダイレクタによる外部サイト誘導

- リダイレクト: 別URLのWebページに転送すること
  - HTTPヘッダのLocationヘッダの設定とか
- 動的に遷移先ページを変更できるWebページの存在
  - 例: ログインしていなくてもある内容を見れるが、必要に応じて、「ログイン(or新規加入)」ボタンを押してログインしてもらうWebページ
    - ログイン処理や新規加入処理の後は元のWebページに戻らせたい

  →リダイレクト先を呼び出し元ページに設定することで実現
- このリダイレクト先を(URLクエリ/POSTなどで)好きなページに設定できたら?

→悪い人が攻撃ページに誘導するのに悪用可能

  - URLを見ただけでは、リダイレクタを備えたWebサービスに見える
  - 普通はサービス関連ページのみリダイレクト可能とする
  - (設定次第で)組織内サービス用のWebページを見れてしまうことも

メールヘッダインジェクション

- ユーザの入力をもとにメールのヘッダ部を生成する構成において発生
- HTTPヘッダインジェクションと同様、改行ができてしまうと、改行後に新たなメールのヘッダ要素を生成できてしまう
  - 例: Subjectの内容に「改行 + Bccで外部へのメール送信」を実現するとか
- かつては「送信先メールアドレスがHTML中に直書き」というひどい実装の話もよくあった
  - 宛先を書き換え放題なのでspam送信に悪用された
  - hidden属性をつければユーザ側から何をやっても見えないと勘違いした人がやらかした

コマンドインジェクション(1/2)

- CGIで入力データをOSコマンドの引数にしている所でサニタイジングを忘れる
  - LinuxベースのWebサーバのCGIでOS(主にUNIX系)のコマンドを呼び出している所でOSやシェルのコマンドを実行
  - WindowsベースのWebサーバでPowerShellを呼び出している所でOSコマンドやPowerShellコマンドを実行
- 入力値に「シェル上の区切り文字を入れた後、実行にするコマンド入力」で成立
  - 例: perlのsystem関数で入力値を引数としたファイルを検索

```
$files = system "/bin/ls $input"
```

  - ここで$inputに「a; /bin/cat /etc/passwd」とすると?

```
$files = system "/bin/ls a; /bin/cat /etc/passwd"
```

    - 「/bin/ls a」と「/bin/cat /etc/passwd」が連続実行される

コマンドインジェクション(2/2)

- マルウェアをダウンロードして実行するコマンドのインジェクションも多い
  - 設置されるマルウェアもバックドアをしかけるタイプが多い
  - テンポラリファイル置き場(どのプログラムも書き込み権限がある場所)にダウンロードして実行させるパターンが多い
    - 通常、CGIはユーザ権限で実行されているので、ファイル書き込み制約
- 他のサーバを操作するコマンドのインジェクションの可能性も
  - 外部からは直接アクセスできないサーバへリクエスト出すとか
- 基本的な対策:
  - 入力された値の範囲のチェック
  - 区切り文字のサニタイジング
  - CGIを実装したプログラミング言語の機能で実装
    - perlでファイル操作にはglob関数とかFind::Fileモジュールとか

URLからの推測

- 他のURLから推測可能なURLに未公開資料を置いてあった
  - 例:「2016/kessan.pdf」をもとに「2017/kessan.pdf(未公開)」を発見される
- Webサーバ側でディレクトリインデクスを有効にしていて、全ファイル名＆下層ディレクトリ名が閲覧可能だった
  - 「2016/」とディレクトリ名でアクセスすると一覧が見えてしまう
  - その中に編集途中で公開時には消したデータを含む物などの都合の悪い物が…
- 対策:
  - 不必要なファイルをWebサーバに置かない
  - ディレクトリインデクスはデフォルトで無効に

ディレクトリトラバーサル(1/2)

- 本来は見れない範囲にあるデータを見ようとする攻撃
- CGIに値を渡すフォーム等のデータを指定部などで、本来見れないないデータを指定できてしまったりする
  - 例: 電子掲示板の書き込み番号など

<input type="hidden" name="datafile" value="file645">

    - hiddenをユーザ側からも見れないと勘違いした非常に初歩的な実装
- 多くのシステムでは".."は1つ上のディレクトリに移動を示す

→入力されても処理しないようにサニタイジングする
- 何らかのミスで処理してしまうと、本来は見れないパスワードやデータの閲覧が可能に
  - /var/www/htmlなどが外部公開の最上位のはずが、それより上のディレクトリも読まれる

ディレクトリトラバーサル(2/2)

- 使っているソフトウェアが汎用の物ならば、データ位置はまず初期名称から変わっていない
  - 例: 某Wikiの初期ディレクトリ/ファイル　　名
  .../wiki/webroot (本来の外部公開の最上位)
  .../wiki/wiki-common (auth.iniとかがある)
  .../wiki/wiki-data
- ルートディレクトリに出るまで..を繰り返して、
  - 例: /etc/hogeの表示を試みる
  ../etc/hoge
  ../../etc/hoge
  ../../../etc/hoge
  以下、繰り返し
- 基本的な対策: 変な値が入力されても良いよう値をチェック
  - あるいは、".."のサニタイジングを入れる

セッションハイジャック

- 認証通過状態IDを盗んで悪用するセッションハイジャック攻撃なる存在
  - (マルウェアを使って)ブラウザのCookie保存ファイルから盗む
    - そこまでやるなら、ブラウザに保存されたID/パスワードを盗むのも…
  - (偽サーバ誘導とからめて)偽サイトに対してCookieを送出させる
  - セッションID固定攻撃をしかけ、事前にクライアント側にしこんだセッションIDが認証通過状態になったら悪用
  - Man in the Middle攻撃(MITM)
    - BASIC認証のような生パスワードを流れる認証も同様
- 対策
  - https下のみCookieを要求するような設定にする
  - 認証通過時に認証IDを再発行

Content Management System (CMS)を狙った攻撃(1/2)

- CMS: コンテンツを登録するだけできれいなページを作成可能なWebアプリケーション
  - データベースに登録されたコンテンツをフォーマットして表示
- 代表的なCMSとシェア[1]
  - WordPressのトップ独走(シェアの50-60%)はここ数年変わらない
  - Joomla, Drupalが一桁%で次点を争っている
- PukiwikiなどのWikiエンジンもCMSの範疇
- EC(Electric Commerce)サイト用のCMSもある
  - EC Cubeがよく使われている印象があると同時に、よく狙われている印象がある

[1] https://w3techs.com/technologies/history_overview/content_management

Content Management System (CMS)を狙った攻撃(2/2)

- 本体だけでなく、プラグインにも脆弱性が見つかり、プラグインから攻撃されることも多々ある
- 対策: PCのアプリケーションと同様、CMSを含むWebアプリケーションのバージョンも最新にする
  - CMSのモジュール、CGIを実行するインタプリタ、Webサーバ用モジュール、バックエンドのデータベースのバージョンも最新にする
  - 自主開発のWebアプリケーションだと、最新にすると動かなくなったりする物が出てくるのがやっかい(obsoleteなライブラリ利用中とかで)
    - 次スライドのWAFの利用も考える
- 最近だとGravのようなデータベースを使わないフラットファイル型CMSの利用も増えている
  - CMS本体は少し複雑になっても、データベースに関する脆弱性の減少の方が大きい?

Web Application Firewall (WAF)

- Webアプリケーションへのインジェクション系の攻撃に特化したファイアウォール
- URLクエリやPOSTの値に、攻撃でよく見られる書式があったらブロック
  - 自分で検知ルールをカスタマイズすることも可能
  - ソフトウェアアップデートが出ていない新規脆弱性に対し、一時的にカスタムルール追加でブロックする運用とかもあり
- 設置形態
  - ネットワーク上に設置するネットワークアプライアンス
  - Webサーバ内にWebサーバと別プロセスとして動作
  - Webサーバソフトウェアにモジュールとして追加

もっと細かく調べたい方

- IPAがすごくしっかりした資料を準備しています

  https://www.ipa.go.jp/security/vuln/index.html#section20
  - 安全なウェブサイトの作り方(2021/3に第7版, 115ページ)
  - 安全なSQLの呼び出し方(2010/3, 40ページ)
  - ウェブ健康診断仕様(2012/12, 30ページ)
  - Web Application Firewall 読本(2011/12に第2版, 98ページ)
  - Web Application Firewallの導入に向けた検討項目(2019/3, 18ページ)
  - TLS暗号設定ガイドライン(2020/7に第3版, 106ページ)
  - DNSキャッシュポイズニング対策(2009/8, 50ページ)
- 変な本を読むよりは、まずはこれを読んだほうが良い
  - ちょっと古い資料はあるが、基礎的な所は変わらないので

概要

- 公開サーバー般に対する攻撃
  - ポート公開中のサービスの脆弱性を狙った攻撃
  - サービス不能(Denial of Service)攻撃
  - サーバ接続のハイジャック
- Webサービスの提供者/利用者への攻撃
  - Webサービス提供者側への攻撃
  - Webサービス利用者側への攻撃
- 認証機構と突破の試み

公開ポートへの認証突破攻撃

- ポートを公開しているサービスには認証が付随する物がある
  - 例: SSH、RDP、VNC、VPNなどの遠隔接続系のサービス
- 基本的な対策
  - (余計なポートを公開しない)
  - (パスワードの強度確保、他との共有禁止、多要素or公開鍵認証)
  - 接続元IPアドレスの制限
  - 時間あたりの認証数の制限
  - 認証失敗回数に応じた無効化(IPアドレス、アカウント)
    - アカウント無効化については、アカウントへのDoS攻撃が成立してしまうので、よほど被害が多い場合の最終手段
    - 最近だと複数のIPアドレスから分散して攻撃してくることも当たり前になってきていてやっかい
- サーバソフトウェアが複雑な設定に未対応ならば、高機能なファイアウォールを導入して設定

Webアプリケーションへの認証突破攻撃

- やられることと対策は公開ポート側と大差は無い
- Webアプリケーション側の認証の脆弱性を突く攻撃もある
- 複雑なことができる分、考慮しないといけない
  - 認証連携でログインやWeb API利用可能としていると
    - 認証連携先が情報流出とかやらかしていないか
    - ユーザが変な権限を認可したりしていないか
- 認証を強化してもセッションハイジャック攻撃側で攻略される可能性もあるので、セッションハイジャックにも注意する

認証突破を狙った攻撃のパターン

- 多いのは、よくあるIDとよくあるパスワードの組での試行
- 昔はID側を固定していろいろ試す攻撃が多かったが、最近では、パスワード側を固定することも
- 「ユーザに他のサービスでのID/パスワードと共用される →他のサービスで流出 →悪用」も多い
  - 管理者から見た場合、一発で認証を突破される形になる
  - 基本的に、ユーザの教育しかない
  - 流出したIDの確認サイトなるものはあるが、そういう物の利用が活発になると、そこに情報売る動きが活発化しそうなのでよろしくなさそう

クライアント側への認証情報窃取攻撃

先のスライドで話したネットワーク上での攻撃で、偽サーバに誘導して認証情報窃取のパターンが多いと思う

- Webアプリケーションへのパスワード認証へのMan-in-the-Middle(MitM)攻撃
  - Web proxyと同等の形で悪性proxy(MitM proxy)は作れてしまう
  - TLSを有効化しているサイトだと、MitM proxy側のSSL証明書に変わる点で判別できるが、そこまで見る人は稀だろう
    - 一部の国家が政府発行のルート証明書のブラウザへのインストールを強制した事例もある[1]

[1] https://gigazine.net/news/20201221-browser-makers-ban-kazakhstan-root-certificate/